|  |  |
| --- | --- |
| **Computer Science Username:** | Bxf03u |
| **Full Name:** | Barnabas Forgo |
| **Student ID:** | 4211949 |
| **University Username:** | psybf |

Important: rename this file by removing the ‘RENAME\_ME\_’ part of the filename before submission. This is to ensure that you submit the correct file.

**G52CPP Coursework Documentation File  
(v1.0)**

(Your chance to tell us what you did and what you want us to give you marks for)

**Each section of this document can be extremely brief! Please just add short bullet points and possibly screenshots, not long explanations.**

**READ ALL OF THE RED TEXT. The red text in each section tells you what to include in that section. You can leave the red text in or delete it, that is entirely up to you, but leaving it in may help you in checking that you did everything.**

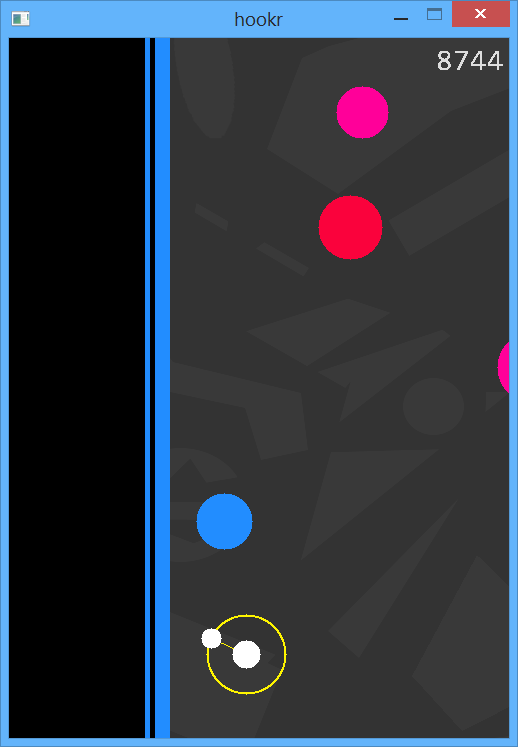
# Overview

Please give a very brief summary (in bullet points) of what your program does. e.g. is it a fast game, a puzzle game, an editor of some kind, a display tool, etc. Basically, you are telling us in general what it SHOULD do, so that we can judge how well it does what it should.

The game I implemented is a fast paced, one-button arcade game. The objective is to get the highest possible score. To do this the player must avoid the obstacles and walls (while unhooked). The game mechanic is explained later.

# Main Screenshot(s)

Please include a screenshot of the main game screen, choosing something which illustrates the game in progress. You can include multiple screenshots to illustrate the game if you wish. A couple of other screenshots will also be included later. Hint: Just use ALT-Print to get copies of the current window for the running program and the paste them into this document.

# Usage

Tell us briefly how to use your program. Do we just run it and it all works or do we need to click on anything on the screen, or press any key to make it do things? i.e. you are telling us how you would like it to be tested. This is your chance to walk us through it if it is complex to get started. In general we will expect your program to just run though.

Just open the Visual Studio project and click start. In the game the instructions are shown. SPACE to start, Esc to quit at any time, m to restart at any time, when the game is over or on the start screen n to show the highscores. The game is only played with SPACE: press to (try) hooking on the closest obstacle, release to unhook. Some pro tips for playing the game:

1. Always keep SPACE pressed, only release for split seconds.
2. The best place to release is when the player is just through the midpoint of two obstacles. This way it will immediately hook on the second obstacle, on a safe trajectory.

# Known problems

Please mention any problems that you know about with your program. If you don’t mention them then we will assume that you do not know about them, and/or that your testing was insufficient.

The trajectory and hook takes the wrong position for a frame or two sometimes. This is probably caused by the hook not getting the updated locations of an obstacle. I don’t think that this issue affects the game a lot.

# Files

Please provide a list of the files which you added, or modified in the following table, along with a brief statement of what they are used for. You may have the .h and .cpp on one line if you wish.

Files which I added/are mine:

|  |  |
| --- | --- |
| **File name(s)** | **Purpose** |
| bg.png | The tile-able background of the game |
| Bxf03uHook.cpp/.h | DisplayableObject subclass, shows trajectory of the player and the hook |
| Bxf03uMain.cpp/.h | BaseEngine subclass, handles events, and dispatches them to the correct object |
| Bxf03uObstacle.cpp/.h | DisplayableObject subclass, shows one obstacle |
| Bxf03uPlayer.cpp/.h | DisplayableObject subclass, shows the player, also indirectly controls the stage |
| Bxf03uStage.cpp/.h | Custom class, converts stage coordinates to screen coordinates, generates new objects, takes over the draw and update loop, handles object creation and deletion, and checks for game over state |
| Bxf03uTileManager.cpp/.h | TileManager subclass, shows the welcome screen, game over screen, highscores, and the “you’ve just beaten” achievements |
| Bxf03uWall.cpp/.h | DisplayableObject subclass, shows the wall on either side |
| consola.ttf | The good old terminal font extracted from windows (hope I’m not breaking any copyrights) |

If you had to change any of the base classes, please provide details of the changes that you made and why. If this was only to do the STL change then you may ignore this, since you will mention his later. If you make a change to the base classes then you need to justify the change and it should work with all sub-classes without modification to the sub-classes. The most common acceptable changes will be generic additions to provide new generic functionality. You should not put code changes or new code which is specific to your program in the framework classes!

Base class files which were modified, and why:

|  |  |
| --- | --- |
| **File name(s)** | **Changes and reasons (i.e. justification)** |
| BaseEngine.h | Changed default window position to centring, made a few draw functions faster by only calculating values for on-screen values, fixed a bug in CopyBackgroundPixels where the game would crash if it is called for off-screen pixels, changing to deque instead of an array |
| BaseEngine.cpp |  |
| DisplayableObject.h |  |
| DisplayableObject.cpp |  |
| FontManager.h | Added #include "header.h" otherwise it wouldn’t compile |
| FontManager.cpp |  |
| JPGImage.h |  |
| JPGImage.cpp |  |
| TileManager.h |  |
| TileManager.cpp |  |
| MovementPosition.h |  |
| Templates.h | Added template based random generator functions, randBetween (based on the good old rand()% method), realRandBetween (uniformly distributed real random) and normalRandBetween (normal distribution random that tries to keep the generated values between min and max) |

# Specific requirements

Consider each of the requirements one at a time and give a brief (bullet-pointed) summary of how you have met the requirement, and why your implementation of it is so good.

If you have failed to implement a feature, or there are known problems with your implementation, then you should include the details here. That way we will know that your testing was not to blame. e.g. if you know something goes wrong under certain circumstances then please say so. Most professional applications have known bugs or problems so this is not a disaster, but you will lose marks for bugs and problems. If you do not mention it here, we have to assume that you thought that the features worked correctly so you will not only lose marks for features which don't work but will also lose marks for not testing it correctly.

For each requirement that you did implement, you should mention how it has been implemented.

This documentation should be in a format which will allow the person marking the coursework to easily identify the various functions (and data members, where applicable) that you added or modified in order to achieve the functionality. **Please be clear and concise rather than wordy. It will be quicker for you to write and quicker for us to read.**

Note: There is no need to explicitly attempt to use specific C++ features if they are unnecessary. E.g. you should not try to alter your program just so that you can put some exception handling in, to prove that you can do so. However, if using a C++ feature is the most appropriate way to handle a problem, you should consider highlighting that you used it (e.g. casting or exception handling) in the relevant sections for the features, or for the complexity or efficiency marking criteria. Your knowledge of these features will be tested in the exam, so I see no need for you to also prove this in the coursework.

Please include a screenshot of the main game screen where appropriate to illustrate your comments, e.g. about your displayable objects or background appearance, choosing something which illustrates the game in progress. You can include multiple screenshots to illustrate the game if you wish.

## Change the framework to use a container class (requirement 1)

**What I did and why I did it this way?**

Insert your text here, justify your decisions (see requirements document) and mention files/functions which you changed.

What container class did you choose? Why?

I used deque. The reason why I chose this is that inserting and removing on both ends is O(1). I thought it would be nice to remove off-screen obstacles from one end in constant time. Then I realised that they are needed until the game session is running as the obstacles should be consistent even if the player turns back for no reason. But I kept it as it has no overhead on common operations (iterating through it, adding objects to the end) compared to a vector which would have been my second choice.

Did you store pointers or objects in your class? Why?

Pointers, as the original array used pointers as well.

Did you make the container class object a global, a class member or something else? Why?

Protected class member to provide extensibility.

Did you keep a pointer to the container class object or the object itself as a variable? Why?

Object as variable, as it doesn’t make a big difference but an object is easier to use and it gets destroyed automatically.

## Implement BaseEngine sub-class object and draw an appropriate background (requirement 2)

**What I did:**

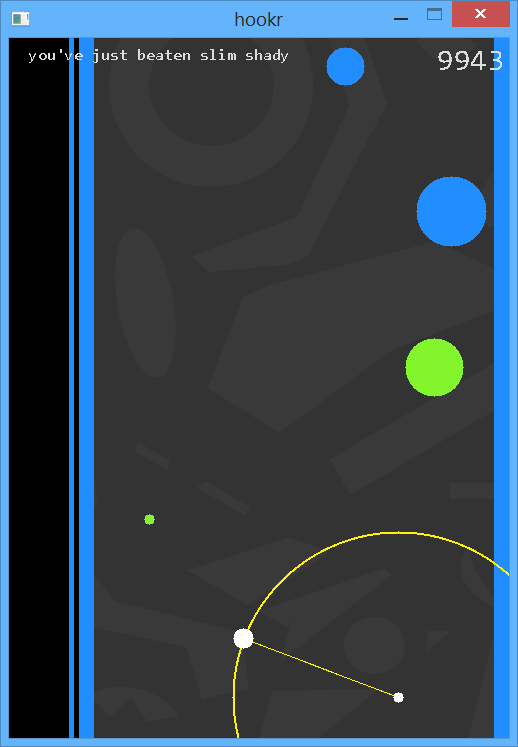
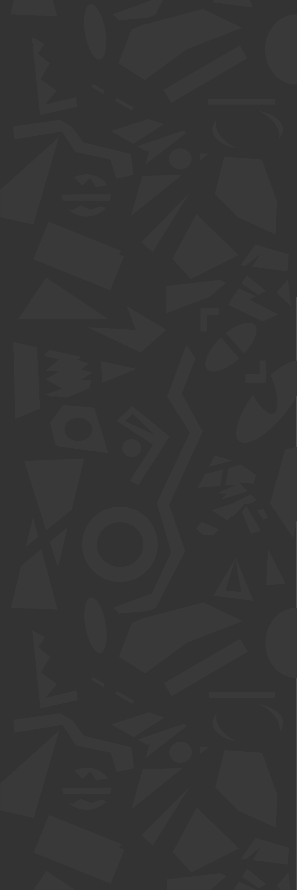
Include any information about your BaseEngine sub-class and the object of that type that you want us to consider in marking.

My BaseEngine subclass (Bxf03uMain) isn’t really interesting; it only keeps track of the game state and delegates events to objects. One thing to note though is that I created a new function to draw rectangles a lot faster than what there was in the BaseEngine. The original function was very slow since every pixel was set one by one on each frame. My solution is to use a quick SDL\_FillRect instead. Since the walls take up most of the screen sometimes this was a crucial efficiency improvement, as with the original function the game’s framerate was cut in half. With my function it was back to its original pace.

Mention any interesting features in your implementation of the background. e.g. what do you think was good that you want us to consider and not to risk missing

As most drawings the background drawing is delegated to the stage object as well. It is a parallax background, that is, it moves with a speed slower than the stage, creating an illusion of depth (or at least I hope so). When the stage is loaded the background image is loaded into memory (using the ImageData class), and is kept there until the game is ended. The image is designed such that it’s top and bottom third is an exact copy so when one end of the image reaches one end of the screen its offset simply needs to be incremented or decremented by 2/3 of the image’s height. The actual image used for the background is shown on the right.

**Screenshot:**

 ****

Please insert a screenshot here illustrating your background

You may include multiple screenshots if you think that it will better illustrate what you have done

## Tile manager class (requirement 3)

**What I did:**

Explain here what your tile manager sub-class does, any complexities in it and any problems that you have (e.g. bugs).

My TileManager subclass (Bxf03uTileManager) is used to display states and keep track of highscores. Each state (started, running, over, highscores) is one “tile” (spanning the whole screen) and the main class passes down the state in the draw loop. It might not be the best choice for this as for most of the variables, the const keyword (on the DrawTileAt function) had to be broken with a mutable keyword to make it work. I originally considered to manage obstacles or the background with the TileManager but it didn’t provide enough flexibility so both was delegated to the stage.

Which class/classes use it?

Only the main class calls the overridden DrawTileAt() to show the states.

What does it do and where is it shown?

It is shown on every screen, the whole of started, over and highscores states are drawn using this class, and the “you’ve just beaten…” achievements are shown from here as well.

What changes the tiles (if anything)?

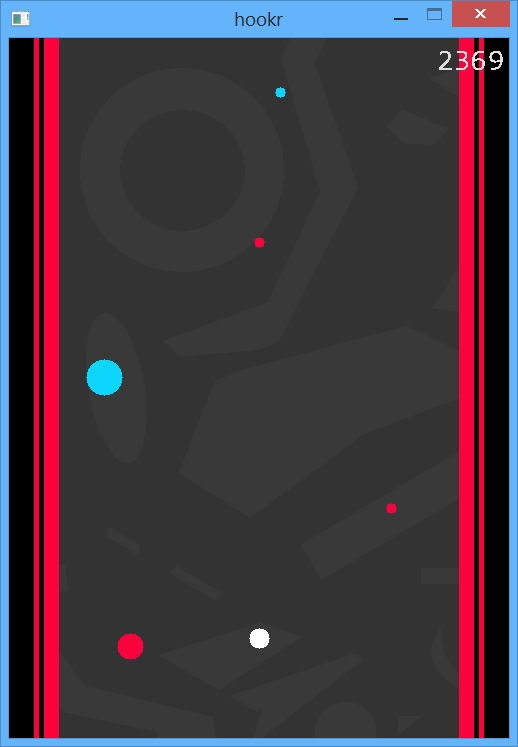
They update themselves, based on information received from the stage and main.

## Player-controlled displayable object (requirements 4 and 5)

**What I did:**

Explain what your player controller displayable object is supposed to do, mention any problems with it (e.g. bugs you know about) and <Insert your text here>

The mechanic is the following: if SPACE is pressed the player object will try to “hook” on the closest obstacle. If it is in front of it, it will wait until the obstacle is either to the left or right of it and only then hook on it. If the obstacle is behind the player it will hook on it immediately. If the player hooked on objects that are in front of the player then the game would be too easy, because if the player is heading towards an object it could just hook on it and avoid collision.



Obstacle behind, immediate hooking.

Obstacle in front, will wait until it gets to the rectangle.

Did you have to do any special coding to make it work well, that you want us to take into consideration?

Describing this movement programmatically was one of the hardest parts. Since this is a very unique type of movement I couldn’t even find other implementations to help me out. I spent a lot of time trying to get the trigonometric calculations right. Then making it move on a framerate independent speed had its own challenges as well.

How does the user/player control it? (Keyboard or mouse? Where is the code to do this?)

With only the SPACE. The keypress is handled by Bxf03uMain and delegated to the player by calling HookOn().

## Second displayable object (requirement 4)

**What I did:**

The requirements involve you creating at least two displayable object classes. Explain your second class, what it does and what the class name is.

If you have multiple additional displayable objects, you may describe each one here

Bxf03uObstacle: class for each obstacle. They maintain a const stage location, and get drawn to the screen by translating those coordinates to screen coordinates. If they are not on screen they are not drawn at all. I had to modify the BaseEngine to make it work since it would crash the program even if one pixel was out of the bounds of the screen.

Bxf03uWall: the walls on the two sides of the stage. I had to implement my own function to draw them since the original DrawRectangle() was too slow.

Bxf03uHook: the yellow hook and trajectory for the player. The DrawHollowOval and DrawLine function had the same issue as the DrawOval, the same modification had to be made.

## Automation (automatically controlled object or decision making, rqmt 6)

**What I did:**

Explain what automatically controlled object (it could be the second displayable object above) or automated decision making you have in your program.

Explain how it works and anything special that you want us to take into account in marking.

All of the objects are automated, since they are moved by the stage, so pick one. I think the most complex is the obstacle, as it manages collision detection, it reacts when it is hooked on and since there are a lot of them efficiency had to be considered as well (ie. not drawing if off-screen).

## Load information from files (half of requirement 7)

**What I did:**

Explain what loading of data from files you have.

The background, scores and the player’s name is loaded from files.

## Save information to files (half of requirement 7)

**What I did:**

Explain what saving of data to files you have.

Highscores are saved to a file whenever one is reached. Handled by Bxf03uTileManager.

## Support different states (requirement 8)

**What I did:**

Explain what code you added and what method you used to create your states.

States are managed by Bxf03uMain and displayed by Bxf03uTileManager. Main uses an enum to give names to the states and the current state is stored in there as well. Using an enum was very convenient as the switch blocks (used in Bxf03uMain::KeyDown and Bxf03uTileManager:: DrawTileAt) were automatically generated for it. The state is modified by the stage and in main by key presses.

**States supported and transition methods:**

List the states that you support and make it clear under what conditions the program will change from each state to each other state.

started

running

over

highscores

n

n

m

m

m

space

collision

**Screenshots:**

Please insert screenshots here illustrating the different states.

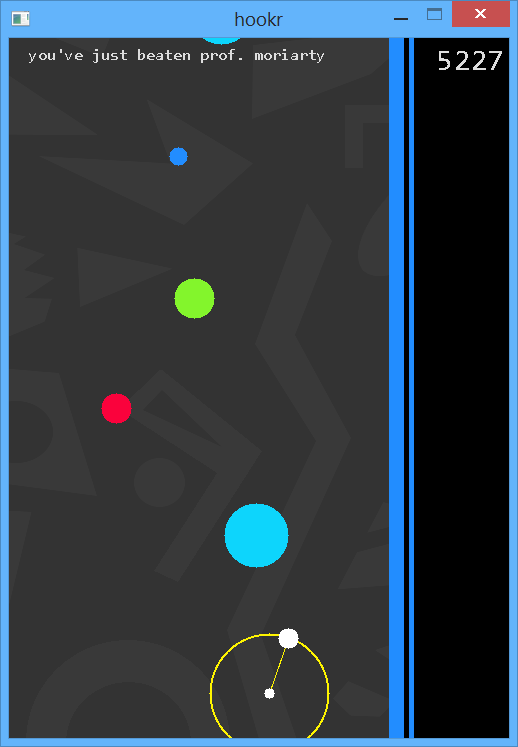
Screenshots are in this order:

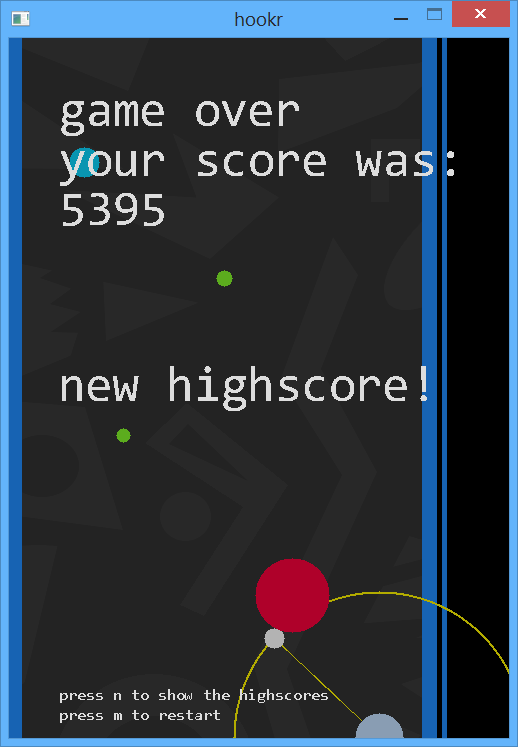
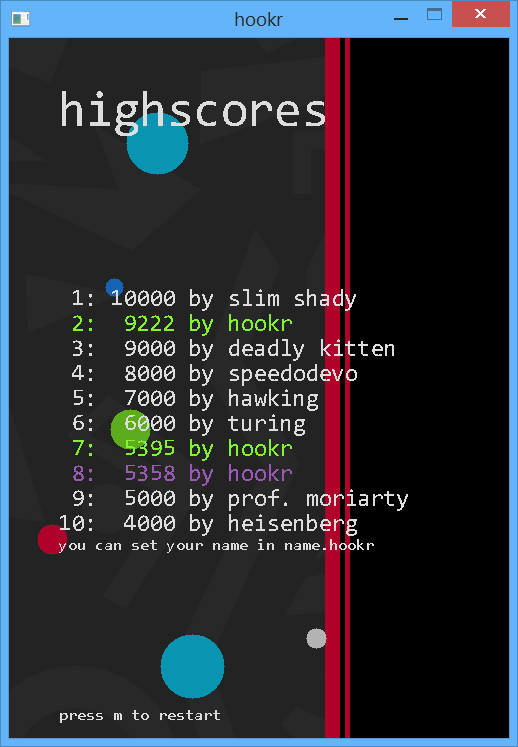
started

running

over

highscores

## Have something react and change (requirement 9)

**What I did:**

What do you have which reacts and changes? Under what circumstance does the change occur? Explain what code you added to do this.

The hook appears and disappears (when hooked/unhooked respectively) and obstacles change colour (when first hooked). These just have a function called by the player when necessary. The walls change colour by checking in the player object if it is hooked. The state changes if a collision occurs. This is checked in the stage (for the walls) and obstacles.

# Marking criteria

Since you know the marking criteria, you may want to make some comments about some of them, to point out something which you would like us to take into account in the marking. E.g. is there a requirement which you think was particularly well implemented?

## Code style and readability

If you want to make any comments about the code style that you selected or to justify any decisions then please do so here.

Do you want to explain anything about your code style of structure?

Maybe you used a consistent naming style which we may not have seen before or maybe you have reasons for what seems to be an odd style? If so then here is your chance to comment.

I didn’t follow the Hungarian notation used in the demos and base classes as I believe that it is outdated, since good IDEs nowadays show the type of variables by just hovering over them and changing the type of a variable also requires to change the name (which can be really hard without an IDE). Furthermore, I think omitting types from variable names improves readability and requires less typing. On the other hand I followed the upper CamelCase naming convention for functions that consist of multiple words and lower camelCase for variables. I commented each logical block of code (where it wasn’t completely obvious what it does) to aid understanding.

## Efficiency

What part(s) of your program do you think are especially efficient or inefficient?

The game rendering can be considered inefficient (when the game is in the running state) since each frame is completely redrawn. This is not a problem for my PC, but it might be for others. On the other hand, there is no easy solution for it since the background and the walls are constantly moving, so the provided facilities in DisplayableObject wouldn’t work. I considered a solution for this though: by keeping a vector of SDL\_Surfaces somewhere (probably in a properly used TileManager), only drawing the obstacles there once (which is very slow) and blitting these surfaces (which is very fast) appropriately on the actualScreen surface. I am not sure if this solution would make the game any faster and I couldn’t test it as I didn’t have the time to implement this.

Other than these I think the game is efficient, as for these reasons I trimmed the computation time as much as possible. For example: BaseEngine modifications that I mentioned before, and (apart from the running state) only requesting a Redraw(true) when necessary. Also I tried to rely on std libraries as much as possible (set of pairs in my TileManager for example), because it is guaranteed that they are more efficient than anything I could come up with.

Are there any justifications that you want to give for things which may appear to be inefficient but are not, explaining why?

Overtaking the Update and Draw loops completely by the stage might seem inefficient, but it was necessary, since the order in which objects need to be drawn is not the same as they need to be updated.

Is there particularly efficient code that you want to highlight to us here?

I think using SDL facilities instead of the BaseEngine is pretty efficient for drawing the walls.

## Robustness, Compilation, Correctness and Reliability

**Known problems with compilation:**

Do you know of any issues with compiling your program? If so then please explain them here.

I don’t know of any compilation problems.

**Known problems at runtime:**

Do you know of any issues with running your program? Does it crash or hang under certain circumstances as far as you know? Does anything not work how it should, or sometimes fail?

As far as I know the only issue is that the trajectory is drawn to the wrong location for a frame or two. I think that this might be because my update order is not perfect, probably the hook is receiving data from the previous update. I stepped through the whole update cycle many times line-by-line, but I couldn’t find the issue.

## Problem/Program Complexity (VERY IMPORTANT)

What do you think was particularly complex about your program?

(You can list multiple things.)

Remember that you get marks for achieving complex tasks, not for writing very complex code to achieve something which should have been simple if you had done it in another way.

I realise that the game itself doesn’t seem to be complex – it is not supposed to be. On the other hand, the implementation underneath it was fairly complex. Figuring out how to move the player on this specific trajectory was especially hard. Then making it work with the scrolling stage was even harder. I hope that this is visible in the code. By the time I had these two working together the rest was fairly simple, although figuring out how to update everything so that each object has the latest update’s data was something I had trouble with and could even figure out perfectly.

## Impact (or appearance)

Is there anything especially good or bad about your program, that should make us go ‘wow’? Is there anything about the appearance of your program that you want to draw our attention to? Please feel free to add screenshots to support your arguments if you wish. Here is your chance to persuade us that your program has the ‘wow factor’.

It is especially important to explain it here if you have some ‘wow’ feature which only happens under specific circumstances.

I’m not a visual designer, so please excuse the background for me. Other than that I tried to make the game so that it has a consistent, nice appearance, despite the framework’s limitations.

# Additional information

In this section you should make any comments which are useful but did not fit the previous sections.

E.g. you may wish to make a comment about additional features which you added which you would like to have considered in marking, or a justification of why your implementation of something was particularly good, but which did not fit one of the above sections.

Alternatively, you may wish to explain any unusual behaviour or problems that your program has which were not mentioned elsewhere, or anything else not mentioned in the previous sections.

Please do NOT mention anything which was already mentioned elsewhere in this document.

Try entering the Konami Code (“wwssadadqe”) while the game is running, then hit m when you get bored. Note that at extra high scores (20000<) the game goes into undefined behaviour range, that is, it wasn’t tested in that range so anything can happen. It can even create a black hole and end human existence as we know it. Although that is very unlikely.