Brandon Wright

[Company name]  [Company address]

Coursework 1 documentation

Table of Contents

[Create an appropriate sub-class of BaseEngine with an appropriate background which is different from the demos 2](#_Toc130043782)

[Create an appropriate new sub-class of the base engine. 2](#_Toc130043783)

[Code snippet (1-5): 2](#_Toc130043784)

[Give it an appropriate background and ensure that the background is different to the demos 3](#_Toc130043785)

[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. 4](#_Toc130043786)

[You MUST create a new class – you must not just change/rename one of the existing demo classes. 4](#_Toc130043787)

[Show your ability to use the drawing functions: 4](#_Toc130043788)

[Draw your background ensuring that you use at least one of the shape drawing functions to draw on the background 4](#_Toc130043789)

[and that you draw at least one image to the background, which is different from the demos and shows your understanding. 5](#_Toc130043790)

[Draw some text on the background 5](#_Toc130043791)

[Draw some text onto the background (not foreground) and ensure that the text is not obliterated when moving objects move over it 5](#_Toc130043792)

[Have some changing text, refreshing/redrawing appropriately which is drawn to the foreground (not background), in front of moving objects 6](#_Toc130043793)

[When the text changes, the old text should be appropriately removed from the screen. 6](#_Toc130043794)

[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 7](#_Toc130043795)

[Provide a user controlled moving object which is a sub-class of DisplayableObject and different to the demos: 8](#_Toc130043796)

[Have a moving object that the user can move around, using either the keyboard OR the mouse (or both) and is different to the demos. 8](#_Toc130043797)

[Ensure that both keyboard and mouse input are handled in some way and do something 9](#_Toc130043798)

[You handle both keyboard input AND mouse input and they both do something 9](#_Toc130043799)

[Mouse 9](#_Toc130043800)

[Keyboard 9](#_Toc130043801)

[Something(s) should visibly change for both – e.g. some position of something or value of something or displayed image, or… 9](#_Toc130043802)

[Provide an automated moving object which is a sub-class of DisplayableObject and different from the one in requirement 5. 10](#_Toc130043803)

[Provide a second moving object (separate to the user-controlled one, with a different class) whose movement is not directly controlled by the player, which moves around, and which looks different to the objects in the demos and to your object in requirement 5. 10](#_Toc130043804)

[Include collision detection between a user-controlled (req. 5) and an automated (req. 7) moving object, so that they interact with each other. 10](#_Toc130043805)

[You need to check for a collision between the two objects. Something should happen when they collide, and something should visibly change – e.g. something moves, direction of travel changes, or something is shown. Collision detection should be at least as good as rectangle-rectangle interaction and should work properly 10](#_Toc130043806)

[Create your own subclass of TileManager. 11](#_Toc130043807)

[Create a subclass of the tile manager which has different behaviour (at least a little) to the demos, is drawn to the background, and is visible to the user when the program is run.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. 11](#_Toc130043808)

[Have at least one moving object interacted correctly with the tile manager, changing a tile: 12](#_Toc130043809)

[Ensure that at least one of your moving objects visibly changes specific tiles when it passes over them – using the position checking appropriately. The tile must be changed and redrawn correctly so that the user sees the change 12](#_Toc130043810)

# 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.

This is seen in **ProjectSrc/cw1\_main.h**. The class psybw7Engine is the sub class of the baseEngine parent class.

### Code snippet (1-5):

#pragma once

#include "../src/BaseEngine.h"

#include "TileMap.h"

class Psybw7Engine: public BaseEngine{

## Give it an appropriate background and ensure that the background is different to the demos

Chart

Description automatically generated with medium confidence

This background is rendered in the **void psybw7Engine::virtSetupBackgroundBuffer()** in cw1\_main.cpp lines(13-23)**.** The following code to render this is (see next page):

What draws the background Shape

Description automatically generated

    int colIndex = 0;

    int colour = 0x478aed;

    int iters = (getWindowHeight()/8)/0x5;

    for(int iY = 0; iY < getWindowHeight(); iY++){

        for(int iX = 0; iX < getWindowWidth(); iX++){

            setBackgroundPixel(iX,iY,colour);

        }

        if(iY % iters == 0){colour += 0x010100;}

    }

This works by setting each individual pixel on the screen and changing the colour of the row using the modulo operator %. I add 1 to the red and green for every modulo returning 0 to increase the gradient based on the height so no matter the height the gradient (should) stay the same. This background is different than the demos. I do not remember seeing any variation on gradients and even if they were some, my method to obtaining this result is way different.

## 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.

The class name can be seen [>>HERE<<](#_Code_snippet_(1-5):) , please refer to this.

## You MUST create a new class – you must not just change/rename one of the existing demo classes.

You can see that I made my own class that is a sub class of the BaseEngine parent class in **cw\_1main.h**. I did inherit functions from the parent class but added my own code such that there is a difference in functionality. I also added my own functions that did not derive from the parent class, called, **drawImage** and **setLabel**. DrawImage works by taking a path, x and y. It then draws the image at the path to an x and y coordinate on the screen.SetLabel is a simple setter function that overwrites the foreground\_label variable.

# 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

A picture containing icon

Description automatically generated

This sun is drawn with the drawBackgroundOval function in the virtdrawbackgroundbuffer function found in the cw1\_main.cpp file on lines 33-39. The lines are drawn with the drawBackgroundThickLine function in the cw1\_main.cpp file found on lines 45-70.

This sun is drawn with the drawBackgroundOval function in the virtdrawbackgroundbuffer function found in the cw1\_main.cpp file on lines 33-39

## and that you draw at least one image to the background, which is different from the demos and shows your understanding.

Icon

Description automatically generated

This is an image of flappy bird being rendered to the background in the file. It is seen in the virtdrawbackgroundbuffer at lines 76-82

# 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

A picture containing shape

Description automatically generatedDiagram

Description automatically generated with medium confidence Figure 1 Figure 2 Figure 3

Chart

Description automatically generated with low confidence

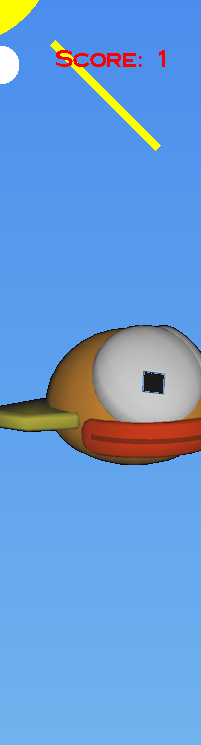
As you can see in figures 1 through 3, a green rectangle object passes over the “Don’t click” text. This shows that an object can move over the text and by figure 3 you can easily see that the text is not obliterated.

The code to obtain this effect can be seen in **cw1\_main.cpp** at line 83. The relevant lines for the moving objects can be seen in **Pipe.h** at line 65 (this is used to move the objects left by adding a negative value to the pipe’s x). The code pertaining to the drawing of the pipes can be seen in **PipePair.cpp** at lines 7-58.

# Have some changing text, refreshing/redrawing appropriately which is drawn to the foreground (not background), in front of moving objects

Figure 1 Figure 2

A screenshot of a computer

Description automatically generated with low confidence 

As you can see with the figures 1 and 2, the text clearly changes over time. This is done by adding to the score and dividing the score by some score debuff metric which forces the score to update more slowly.

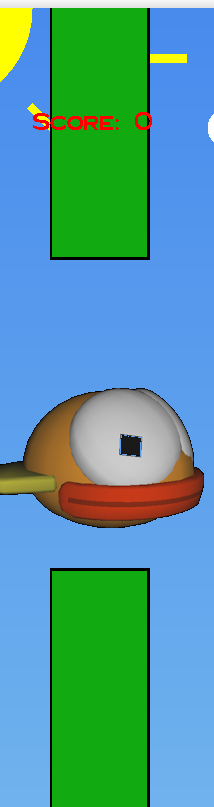
## When the text changes, the old text should be appropriately removed from the screen.

See [>>Here<<](#_The_text_has)

## 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

Figure 1 Figure 2 Figure 3

A picture containing logo

Description automatically generated  A screenshot of a computer

Description automatically generated with medium confidence

As you can see in figures through 3, a green rectangle moves under the “Score: 0 text”. This shows that the moving objects move UNDER the text rather than on top of it.

# 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) and is different to the demos.

The player object, the bird, is controlled with the space bar. This bird ONLY moves on the y axis, however it would be very simple to implement code to move the bird on the x axis. The reason for the bird not moving on the x axis is that this project is a recreation of the game flappy bird, in which the bird only moves on the y axis to avoid pipes. The bird is controlled via gravity + velocity in which gravity is constantly added to the velocity to increase player speed downward and the velocity is added to the y position to move the player. There is a velocity limit so the player doesn’t infinitely accelerate. The code for this can be seen in **Bird.h/Bird.cpp** (the entire file is pretty much handling all the stuff for movement and control)

NOTE: the bird mentioned is different than the background image shown earlier in examples/justification on requirements. It is the bird in the following image:

A picture containing text, traffic light, transport, aircraft

Description automatically generated

# Ensure that both keyboard and mouse input are handled in some way and do something

## You handle both keyboard input AND mouse input and they both do something

### Mouse

The mouse handles one thing, and that is drawing new text to the screen. It also has debugging capabilities but to the user that is only seeing the application and not the console window, this would be redundant. The code for drawing new text to the screen can be seen in **cw1-main.cpp** at lines 168-170

### Keyboard

See [>>Here<<](#_Have_a_moving)

## Something(s) should visibly change for both – e.g. some position of something or value of something or displayed image, or…

Player moves – keyboard

Next text rendered to the screen - mouse

# Provide an automated moving object which is a sub-class of DisplayableObject and different from the one in requirement 5.

## Provide a second moving object (separate to the user-controlled one, with a different class) whose movement is not directly controlled by the player, which moves around, and which looks different to the objects in the demos and to your object in requirement 5.

Chart

Description automatically generated

These rectangles are the games “pipes”. The x position for these are found within the **Pipe.h** file and the functions to move them is found on line 65. The code that actually accesses this X value and manipulates it is found in **PipePair.cpp** in lines 67-75.

# Include collision detection between a user-controlled (req. 5) and an automated (req. 7) moving object, so that they interact with each other.

## You need to check for a collision between the two objects. Something should happen when they collide, and something should visibly change – e.g. something moves, direction of travel changes, or something is shown. Collision detection should be at least as good as rectangle-rectangle interaction and should work properly

The bird class, seen in **Bird.h** has a function called **checkCollisions**, which handles all of the collision detection. This works by the bird grabbing each pipe from the displayable object array and running the collision detection on each of them. By this I mean bird vs pipe. So the bird checks if it is intersecting with every pipe per frame. The code for the collision detection itself is seen in **Bird.cpp** at lines 46-101. This collision detection uses the UtilCollision class’s static function called checkRectangles. Due to how weird the collision code was made, i.e you draw rectangles via width and height but check collisions via corners, I had to do some fiddling about with magic numbers on the correct values pertaining to the bird and pipes. When the collision detection happens, the score is reset and the bird is set to a predefined x and y position (this does keep the bird stuck in a pipe till it has passed over the bird though)

# Create your own subclass of TileManager.

## Create a subclass of the tile manager which has different behaviour (at least a little) to the demos, is drawn to the background, and is visible to the user when the program is run.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.

This is done in the file **TileMap.h**. The tilemap is named Psybw7TileManager. The usage of this tilemanager is to draw the clouds on the background, seen here:

**Icon

Description automatically generated**

The code for drawing clouds can be seen inTi**ileMap.cpp** on lines 18-43.

And it is used to animate the tiles behind the player to show how high they are as a form of a meter. The tiles to show how high the player is are yellow, the others are the base colour of the background. Seen here:

Chart, bubble chart

Description automatically generated with medium confidence

The tiles the player has least recently occupied.

The tiles the player has most recently occupied.

# Have at least one moving object interacted correctly with the tile manager, changing a tile:

## Ensure that at least one of your moving objects visibly changes specific tiles when it passes over them – using the position checking appropriately. The tile must be changed and redrawn correctly so that the user sees the change

See [>>HERE<<](#_Create_a_subclass)