Game Report

A simple flash game…

…By Samuel Bowen

Contents

Table of Contents

[Game Summery 5](#_Toc408442531)

[Objectives 5](#_Toc408442532)

[Main Objective 5](#_Toc408442533)

[Sub Objectives 5](#_Toc408442534)

[Rules 5](#_Toc408442535)

[Constitutive 5](#_Toc408442536)

[Implicit 5](#_Toc408442537)

[Operational 5](#_Toc408442538)

[Gameplay 5](#_Toc408442539)

[Game Structure 7](#_Toc408442540)

[Screens 7](#_Toc408442541)

[Main Menu 7](#_Toc408442542)

[HighScores: 7](#_Toc408442543)

[Difficulties Page 7](#_Toc408442544)

[Game Level/Map & Scoreboard 7](#_Toc408442545)

[Defeat Screen 8](#_Toc408442546)

[Implementation Specification 8](#_Toc408442547)

[Main Code 8](#_Toc408442548)

[Variables 8](#_Toc408442549)

[Button functions 9](#_Toc408442550)

[Main Setup Function 9](#_Toc408442551)

[Game Event Handler Function 10](#_Toc408442552)

[Mouse Event Functions 10](#_Toc408442553)

[Navigation functions 11](#_Toc408442554)

[Clean Game Function 11](#_Toc408442555)

[Classes 11](#_Toc408442556)

[BackGround 11](#_Toc408442557)

[Bullet 11](#_Toc408442558)

[BulletManager 12](#_Toc408442559)

[eBulletManager 12](#_Toc408442560)

[Enemy 13](#_Toc408442561)

[Enemy2 13](#_Toc408442562)

[EnemyManager 13](#_Toc408442563)

[HighScores 14](#_Toc408442564)

[Item 14](#_Toc408442565)

[healthCrate 14](#_Toc408442566)

[ItemManager 14](#_Toc408442567)

[ScoreBoard 15](#_Toc408442568)

[Player 15](#_Toc408442569)

[Critical review 16](#_Toc408442570)

[Appendix 1 19](#_Toc408442571)

[Installation guide - Desktop 19](#_Toc408442572)

[Installation guide – Android 19](#_Toc408442573)

[Appendix 2 19](#_Toc408442574)

# Game Summery

## Objectives

### Main Objective

Acquire the highest score possible.

### Sub Objectives

Killing enemies – this will rewards points towards the user’s score,

Collecting score crates – Some crates will reward points.

Surviving will allow for more time to collect points:-

Avoiding damage from enemy is the best way to survive,

Collecting health crates – Some crates will contain health and restore health to the user.

## Rules

### Constitutive

The user’s score will increase as they kill enemies or collect score crates.

The game will end if any of these conditions are met:

* Time runs out
* The user’s life reaches 0

### Implicit

Do not abuse bugs.

### Operational

The user can only move left, right and jump using either the following keys:  
a(left), w(jump), d(right), also the left, right and up arrow keys for keyboard input.

The 3 buttons displayed on screen are touch enabled to allow touch movement, located in the bottom left of the screen.

The user can shoot using the mouse click/ touch screen almost anywhere on the screen, there is a dead zone where the movement buttons are located on the screen which will not cause the player to shoot.

The user’s score will increase when they kills an enemy or when he picks up a score crate.

The user’s health will decrease when they are hit by an enemy unit, or hit by an enemy’s bullet.

The user’s health will increase when they pick up a health crate.

## Gameplay

The user starts on the main menu screen, on that screen there are three buttons; play, highscores and quit. Quit will quit the application, highscores will take you to the highscores page and play will bring you to the difficulties page.

In the highscores page the top 3 highscores will be displayed and a back button that will allow the user to return to the main menu.

The difficulties page presents another 3 buttons labelled; easy, medium and hard. These buttons represent the different difficulties of the game and will begin the game at their respective levels when clicked.

When a difficulty has been selected the player will enter the game and be presented with 3 movement keys, they each visualize and arrow in which the player will move if they are clicked. The player can also see their current score, their health and the time remaining. The player is able to shoot by clicking/touching the screen.

When the time or health reaches 0 the player will then be taken to the highscores page, but this time it will also display the score the player was able to achieve. They are once again presented with a back button to return to the main menu.

# Game Structure

## Screens

Static Text – Title.

### Main Menu

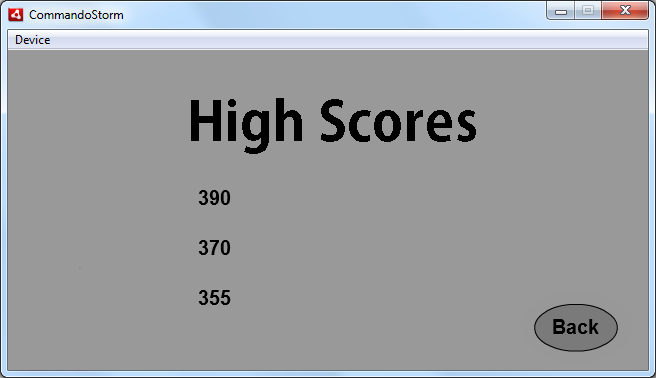


Button - Takes the user to the difficulties page.

Button - Takes the user to the high scores page.

Button - Quits the game.

### HighScores:



Static Text – Title.

Dynamic Text fields - Top 3 high scores.

Button - Takes the user back to the main menu.

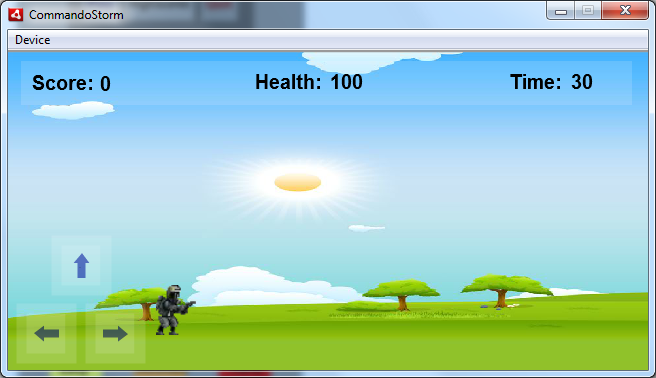
### Difficulties Page



Static & dynamic text fields - User’s score.

Buttons - Easy, medium and hard difficulties.

Game Level/Map & Scoreboard

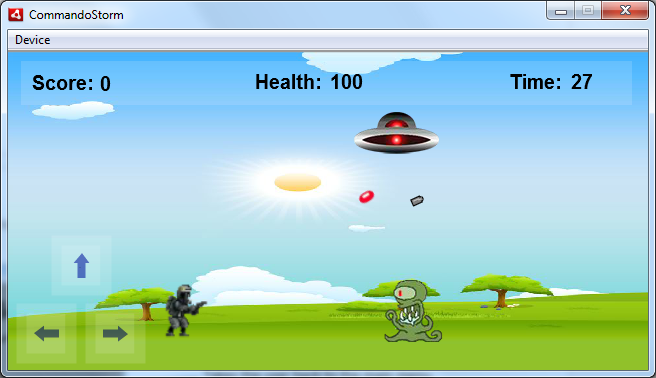


Buttons - Movement touchscreen keys.

Static & dynamic text fields Time remaining

Static & dynamic text fields - User’s health.

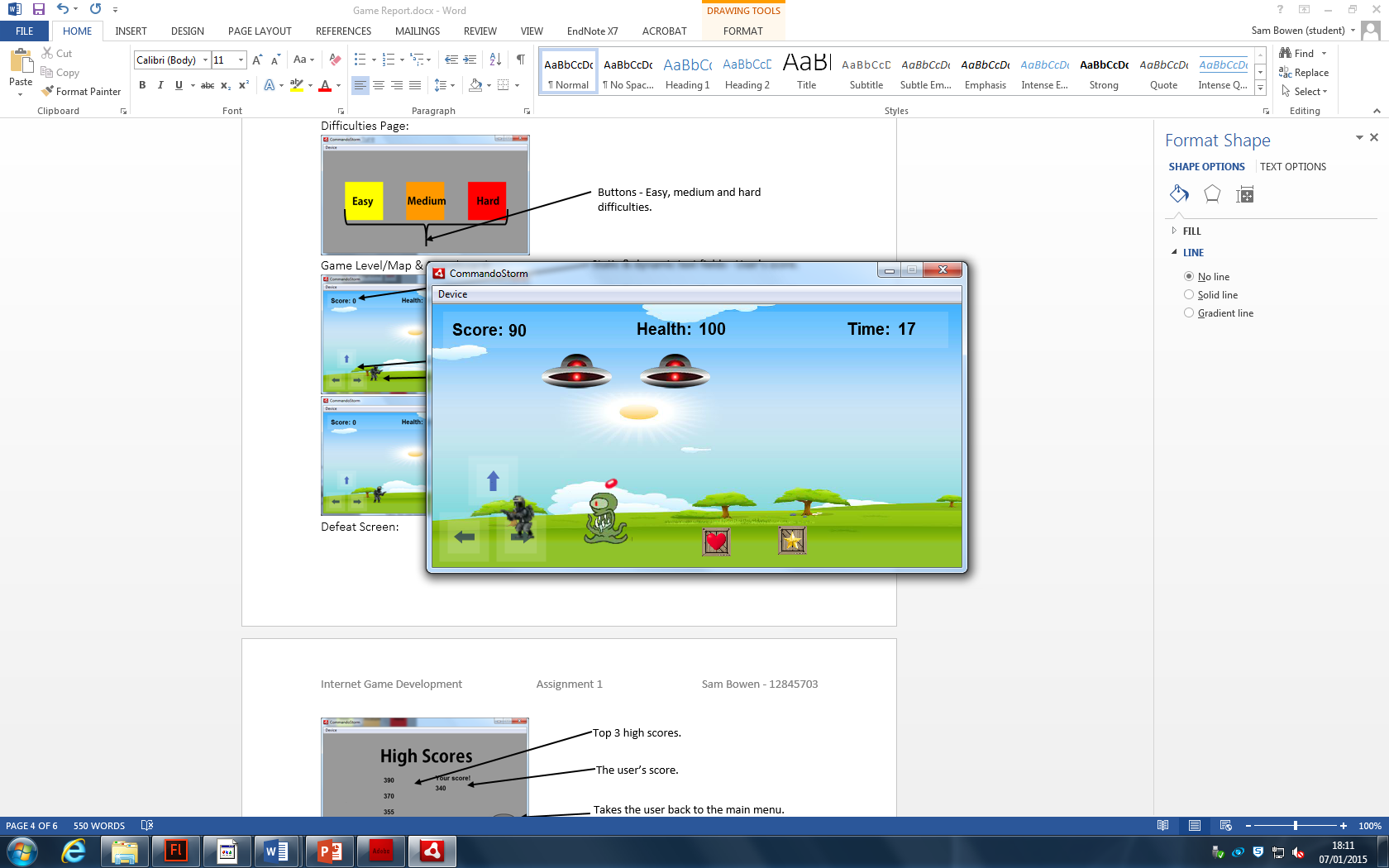
Movie Clip - Player.



Movie Clip - Enemy unit1. Animated, consisting of 5 image frames which play while it moves.

Movie Clips - Player Projectile (Grey), enemy projectile (red).

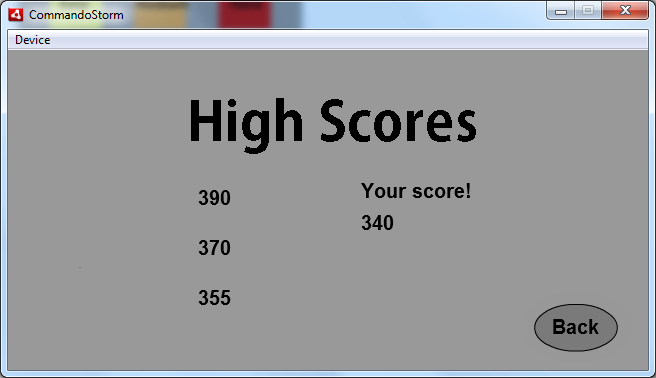
Movie Clip - Enemy unit2.



Movie Clip – Score crate

Movie Clip – Health crate

### Defeat Screen



Dynamic Text - The user’s score.

Dynamic Text - Top 3 high scores.

Button - Takes the user back to the main menu.

# Implementation Specification

The body of my code is located within frame 1 of the actions tab. There is only one scene in my game and everything is done on the first frame.

## Main Code

### Variables

The actions tab contains some of the fundamental game settings as constant variables; game time, player fire rate and enemy fire rate. The fire rates are used within the main game loop to control how fast the player and enemies shoot. The game time is also used within the game loop along with a counter to provide a time limit to the game, this is then updated and displayed on screen via dynamic text in the scoreboard class.

Controlling the difficulty is done with a simple variable within the main game, it is changed by clicking on the different difficulty buttons. Then when setup is called and the enemy manager is added to the scene it will change the statistics of the enemies by passing the difficulty variable.

I used managers to create/update/destroy groups of objects. In my game I am using 4 managers; Enemy manager, player bullet manager, enemy bullet manager, item manager. The enemy manager looks after the two different types of enemies, while the item manager also looks after two different types of items. The bullet managers are self-explanatory.

I have added sound to my game and have done so by importing sound files into the library and then given them their own classes which extend “sound”. The class MyFavSong has been created for my background music and plays on repeat throughout the game. It’s been converted into a sound channel so I can track when it ends to replay it. There are two other sounds for the player bullets and enemy bullets.

I also have some other variables which have specific uses such as firing\_bool, currentScore, currentX and currentY. Firing\_bool tracks if the player is holding down the fire key, currentScore holds the player’s current score to pass into the highscore page and current, currentY hold the last known position of the mouse.

### Button functions

Within the main actions code base there are quite a few buttons functions which are pretty much self-explanatory and help the user navigate through the game.

Start button – this will hide the main menu and display the different difficulty buttons by calling the hideMainMenu and showDifficultyButtons functions.

Difficulty buttons – these buttons will change the difficulty of the game by changing the difficulty variable and then hide themselves. After that they call the setup function.

Quit button – this will call the fscommand “quit” which closes the flash application, and NatveApplication.nativeAplication.exit() to close android apps by calling the quitGame function.

Back button – this button is located on the highscores page so it will hide the highscores page and then call the replayGame function which will then display the main menu by calling the showMainMenu function.

Highscore button – this button will hide the main menu and then display the highscores page which consists of the top 3 highscores it does this by calling the hideMainMenu function and showHighScores function from within the highscores class. It also seperatly makes the back button, and title visible.

### Main Setup Function

This function is only called by the difficulty buttons, it is used to initialize almost all of the game. The functions begins by resetting the time to the set constant variable. It then initializes the scoreboard using its init function and sets the time text field to the current time left in the game, it then sets the interval for the counter to make sure the timer is counting down in seconds (1000) milliseconds.

Once the scoreboard and timer has been set up the function will then initialize all of the managers, these managers are added as children to the scene and are created using the classes constructors, most of them will pass in parameters to help with their functionality.

The highscore class is then initialized, along with the player class, the item manager is passed through to the player for collision detection. Player shooting variables are then reset and the event listeners for the mouse are added. The touch movement buttons become visible for the user to see and use and the background initialized.

The last piece of code which is in the setup function is the event listener which is the main function within the game.

### Game Event Handler Function

This function is the main game loop which will constantly be called.

Within this function I have caused the background music to continuously loop using a sound channel function called position, using position I worked out the end of the song and when the song reaches the end it will replay.

Within this function there is an if statement to regulate the rate at which the user can fire his weapon. This if statement also controls where the player is shooting too by returning the position of the mouse when fired, as well as flipping the characters image if the player attempts to shoot backwards.

The if statement checks if the player is current shooting, by checking the firing\_bool which is only true if the mouse click is being held down or the screen is being tapped. The other condition is if the enough time has passed since the last shot. If all of these conditions are met a bullet is created through the bullet manager, the bullet firing sound is played and the timer is reset.

There is another if statement which is very similar which causes the enemies to fire on a timer, this will cause the enemies to all fire at the same time by using the enemy manager to create bullets for each enemy. This also plays the enemy fire sound and resets its respective counter.

There are also checks against the player’s position to see if the sprite should be moving across the screen of whether the background should be moving behind the sprite.

This function will then call all of the move functions within the managers, this causes the positions of all of the objects on the screen to update this x and y positions.

The function will then do all of the collision checking associated with the objects, each object manager has a collision detection with another object or object manager.

The games end conditions are then checked at the end of this loop function, an if statement to see if the player’s health has been reduced to 0 is called and also an if statement to check to see if the time has been reduced to 0 is called. If either of these are true then the show score function is called to show all of the current top highscores and the score you achieved.

### Mouse Event Functions

Mouse Down – This function is called when the left mouse click is pushed down, incidentally it also works on android for the touch screen. Splitting the left mouse key into up and down states I am able to allow the player to auto fire if the key is constantly held down. I do this by using my firing\_bool.

Mouse Up – This function is called when the left mouse click, or the screen is released. It will set the firing\_bool back to false to stop the user from firing.

### Navigation functions

My navigation functions work by setting the visible properties to true and false.

Hide Main Menu – hide the main menu.

Show Main Menu – displays the main menu.

Hide Difficulty Buttons – hides the difficulty buttons.

Show Difficulty Buttons – displays the difficulty buttons.

Hide Highscore Menu – hides the highscore menu.

Show Highscore Menu – displays the highscore menu.

Hide Movement Buttons – hides the movement buttons.

Show Movement Buttons – displays the movement buttons.

### Clean Game Function

This function is used to clean the game and try to prevent memory leaks by removing all managers, event listeners, timers and other objects. It is called just before the highscore screen is displayer after a player has completed a run through of the game. It utilizes the destroy function within the class managers to destroy all of their children as well as the managers itself. It also calls remover event listeners to stop the program looping though the game event handler function. It is also used to reset the position of the player along with the score, time and health scoreboard properties.

The quit game function is there to call the fs command “quit” and NativeApplication command quit to attempt to close down the game.

The show score function is used to compare the player’s current score against all of the currently stored highscores, it then takes the player to the highscore screen.

Finally I have added all of the event listeners for the buttons and the touch screen button movement functions.

## Classes

### BackGround

This class extends a movie clip, background is currently the image of the background I am displaying in the game. This class is used to move the background left and right (long the x axis) to simulate player movement when he reaches certain points. It also has a reset function to set it back to its starting point and an init function to make the background visible.

### Bullet

This class contains variables which have been hard coded to hold the size of the screen, which allows the bullets to know when they should be deleted. It also holds the speed at which the bullets should move and the distance between the bullet and its target.

The bullet class contains two main functions, the constructor and the movement. The constructor class takes four parameters, these are the coordinates of the object that wants to shoot the bullet and coordinates that the bullet should travel to. These values are stored in private variables inside of the class. The constructor also works out the distance between the target and the position it was shot from.

The move bullet function is used to move the bullet’s x and y coordinates in the direction of its target position. I have managed to achieve correct sprite rotation and bullet speeds by using trigonometry to figure out the angles between the target and the firing position. If the bullet moves off of the screen it will destroy itself.

The destroy function simply removes the child from the parent.

eBullet – Enemy bullet, identical to the other bullet class but the movie clip contains a different sprite image.

### BulletManager

This class is used to manage all of the player’s bullets at the same time. It contains 3 variables, two are references to other classes and one contains the damage the bullet will deal. The bullet manager class needs to access the scoreboard to update the player’s score when they kill an enemy, and the other is to spawn items when the enemies die.

The bullet manage constructor simply takes two parameters of type scoreboard and item manager which allows the use of their functions.

The create bullet function takes in four parameters just like in the bullet class, once again they are the coordinates of where the bullets are shot from and their target. The bullet manager then creates an instance of a bullet and gives it these four parameters to pass to the bullet constructor. The bullet manager then adds this instance of the bullet as a child.

The move bullets function simply loops through all of the bullets that this class has and moves them one by one by creating instances of each of them. They are moved using the bullet classes’ move function.

The check collision function checks for collisions against an instance of an enemy manager, this is because these bullets are they players and are looking to collide with enemies. This function needs to create multiple instances of temporary objects to check for collisions against. First off it will loop through each of the bullet children this class contains, a new temporary bullet is created each time, within that loop it will then loop through each of the enemies within the enemy manager class, once again a temporary enemy is created for each enemy. This is where the function needs to see which type of enemy the bullet is trying to collide against, as there are two different enemy classes, enemy and enemy2, it does this by asking if the child of the enemy manager at this point in the loop is of type “Enemy” or “Enemy2”. The function then uses the built in flash function hit test object to check to see if the two object’s sprite’s bounding boxes collide with each other. It this returns true then the enemy’s health is reduced by the damage of the bullet, the bullet is destroyed and another check is issued to see if the enemy has suffered a fatal blow. If the enemy has died the player’s score is increased by the amount of points that particular enemy rewards, and depending on the enemy it will attempt to spawn an item using the random function to give it a percentage chance to spawn.

eBulletManager – Enemy bullet manager, almost identical to the normal bullet manager, this one instead of creating the bullets from the player’s coordinates to the mouse coordinates it will create the bullet onto of the specific enemy and fire the bullet towards to player. The constructor takes the scoreboard so the health of the player may be reduced if it is hit by a bullet.

The create bullet function uses the temporary enemy’s coordinates to give the bullet its position on the screen.

The move bullets function is identical to the bullet manager’s move bullets function.

The check collisions function needs to check for collisions against the player, so instead of having two loops, one for the bullets and one for the enemies it only needs to check against each bullet and the player. This is done the same way as the other check collision’s function using temporary bullets. If the player is hit but a bullet the bullet will be destroyed and the player will lose health equal to the damage of the bullet.

### Enemy

This class extends a movie clip, it is also linked to a sprite sheet of an alien enemy that plays though when it has been created. The class contains multiple variables which effect the characteristics of the enemy, its speed, damage, health, and points are held as Number/integer variables. This class also contains a hit counter variable to stop the player from being hit the same enemy multiple times.

The constructor does nothing but build the object.

The initialize function takes 4 parameters, the x and y coordinates of the enemy, its “strength” and the stage. The coordinates determined where it will be spawns on the screen, where the strength variable is used to alter the enemies’ variables to change the difficulty of the game. Each enemy’s speed is set at random when they are created.

The movement function for enemy one is simple, they will constantly move to the left until they move off screen where they will be destroyed.

The damage player function will return the damage the enemy will inflict upon the player, the hit player function will set the enemies hit counter to true if they have already hit the player once, to stop the player from losing too much health from one enemy.

The reduce health function will reduce the health of the enemy by the amount of damage passed in as the parameter, if the unit’s health falls to 0 or bellow it will be destroyed.

The get health function returns the health of the enemy, and the destroy function simply destroys it.

### Enemy2

The Enemy2 class is almost identical to the Enemy class, but there are a few differences, firstly the enemy2 class uses a different sprite, it also has different values set to its variables which makes it have different health, and damage values.

The move enemy function is also different, I do not want these enemies to move off of the screen, so the function will make sure the enemy doesn’t go past a certain point and will cause it to move in the opposite direction.

### EnemyManager

The enemy manager class extends a movie clip and is used to handle all of the enemies at the same time. It is very similar to the bullet manager class in the fact that it create and moves enemies in the same way, except that it creates two different types of enemies at different heights. Its collision detection function is almost identical to that of the enemy bullet manager’s as they are both checking against the player. The enemy manager’s main difference is that the enemies are spawning on a timer instead of a key press, this is done by using the timer class and an enemy frequency constant variable.

### HighScores

The highscore class extends the type movie clip, it holds three shared objects that contain the top 3 highscores on the current machine.

In the highscore constructor function the text fields on the highscore timeline are given the current highscores. There are two functions to hide and display these three text fields on the screen by setting their visible properties to true or false.

The compare high score function takes one parameter which is the player’s current score, it then checks it again the lowest of the 3 highscores and returns true if the player has *at least* beaten the lowest highscore and false if they haven’t.

The update highscore function uses a simple bubble sort macro to move the highscores into the right order, it will then also update all of the highscores and save the shared objects and then update the text fields with the new highscores.

### Item

The item class extends movie clip and is linked to its own unique sprite, it also consists of 3 variables, two of which are used to calculate the item’s expiration speed, the other is the amount of score the player will be awarded for picking it up.

The item constructor takes 3 parameters, two are used for its x and y coordinates and the other is to set the score that the item will grant the player.

The items has a move item function which moves it down the screen until it hits the floor, this is because some items will spawn higher up on the screen and need to be accessible for the player. The move function also handles the life time that the item has, every time it moves the time counter will increase and when the timer reaches the limit the item will be destroyed.

The item also has a move right and left function which are called when the background needs to move to the right and left as well to simulate the item moving when the background moves.

A get score function is there to return the amount of points the item will give to the player.

### healthCrate

The health crate class is almost identical to the item class, it does everything the same as the item class but instead of increasing the player’s score it will increase the player’s health. It also extends movie clip and has its own unique sprite.

### ItemManager

The item manager class is the same as the other manager classes, it extends type movie clip and is used to manage all of the items and health crates at the same time.

The item manager class holds the variables that the item and health crates will alter the player’s score and health by, it also uses functions from the scoreboard class and contains an instance of the class scoreboard.

The constructor has one parameter which is of type scoreboard and then sets up the instance of the score board.

It has two creation functions which create the item class children and the health crate class children, just like the other managers. It has two parameters, the x and y coordinates of the two items.

The move items function works the same as the other managers by looping through all of the children that the manager has and calls each of the move functions linked to the items.

Move items left and right are used to move the items along with the background, due to poor programming I had to use somewhat hack code to get the background to move when the player hit a wall.

The check collision function works the same as the enemy manager’s check collision function as it is checking for collisions against the player. When the item class object collides with a player and the hit test object function returns true it will call the scoreboard function increase score to increase the player’s score. If the health crate class object collides with a player the player’s health will increase using the scoreboard function.

### ScoreBoard

The scoreboard class extends movie clip and contains 6 text fields, two are dynamic text fields changed by the class, 3 are static labels and one is changed by the main game. The scoreboard class holds the player’s health and score, is uses two variables, score and health to do this.

The scoreboard constructor calls reset scoreboard to hide the scoreboard by turning all of the text field’s visible properties to false.

The scoreboard initialize function resets the score and health variables to a set value, and then update the dynamic text fields which hold them. It also displays the scoreboard by turning all of the text field’s visible properties to true.

Reduce health, increase score and increase health are all functions that take one parameter, an integer equal to the amount the function needs to increase/decrease by. These functions also update the text fields after they have updated their respectable variables contents.

The get score and get health functions simply return the score and health variables respectively.

Reset scoreboard is used to hide the scoreboard, and time text is used to update the dynamic text field which holds the time limit.

### Player

The player class extends movie clip and has its own sprite linked to it. It also contains a lot of variables which are used to enable player movement, the other variables are used to set a “floor” level, speed limit and maximum jump height.

The constructor sets the player’s visible property to false. The initialize function takes one parameter, an instance of item manager, which is used to move the items left and right along with the background.

The initialize adds event listeners for the player which will listen for keyboard input from the user. It also sets some variables to be equal to the current position of the player and turn’s the player’s visible property to true. The positions player function is also called.

The position player function sets the player into its default position by changing the player’s x and y coordinates.

Reset player is used to reset the player’s position as well as removing all of the event listeners from the player class.

Check keys up and keys down are called on keyboard events, they check for specific key codes which represent the “a”, “w”, “e”, “left”, “up”, and “right” keys on the keyboard. If any of those keys are pressed then their respectable move/not move functions are called to move the player.

The move functions set Boolean variables to true/false depending on the input from the keyboard.

The enter frame event handler is the function which handles all of the movement for the player, it uses all of the Boolean variables to determine if the player object should move on the screen or not. It does this by checking the player’s current position and movement Booleans. If the player object is not touching one of the designated walls then the object will move left or right, but if the object is touching one of the walls then the background and will move in its place.

The jump function I have used from someone else’s code which has been referenced at the bottom of the report.

# Critical review

I believe that the code is efficient, functions have been created to eliminate the need to constantly reuse the same lines of code and have a simple function call in their place.

All functions have also been created using the minimal amount of code possible to try and keep the code precise and effective.

The overall game concept is good, the idea of an extremely repeatable game is quite appealing and the implementation on desktop is quite smooth.

The game in its current state lacks content, even though the concept is good there is not enough stuff to do within the game to make it a good game. By adding extra features to the game such as different weapons, or more enemies would make the game much better.

The game lacks lots of animation, it only has one animation in the game which is the alien’s movement, it would be much better if the game was more interactive and included animation for all of the other objects within the game, and gave each object multiple animations such as movement, death and shooting animations.

The game could use much better code when being ported to android, currently multiple features are being lost when the game is being created into an APK and played on android. This would be fixed by fixing the code itself.

References

Images

Commando Character sprite - <http://theen4cer.deviantart.com/art/Sprite-Wars-Commando-Temp-79946188>

Alien character – Created in photoshop - Based off of the simpson’s Kang and Kodos - <http://simpsonswiki.com/wiki/Kodos_Johnson>

Spaceship – Created in photoshop

Player & enemy bullets – Created in photoshop

Plain Crate - <http://opengameart.org/content/2d-wooden-box>

Heart Crate – Created in photoshop using the Plain Crate & a heart sprite - <http://www.clipartbest.com/free-heart-image>

Star Crate – Created in photoshop using the Plain Crate & a star sprite - <http://pngimg.com/img/objects/star>

Background - [hdw.eweb4.com](http://hdw.eweb4.com/out/1034630.html)

Code

Movement – To begin with I learnt how to program in flash using this handy website, I expanded upon some of their code and have also used some of it. Here’s the movement functions - <http://www.flashgametuts.com/tutorials/as3/how-to-create-a-platform-game-in-as3-part-1/>

Most of the other code I have learnt from is from your moonflyer game. I have mostly expanded on your code and then written my own.

Game Concept

My game idea is based off of the miniclip game Commando.

# Appendix 1

## Installation guide - Desktop

1. Download CommandoStorm flash file and all ActionScript 3 files.
   1. BackGround.as, Bullet.as, BullerManager.as, eBullet.as, eBulletManager.as, eBulletSound.as, Enemy.as, Enemy2.as, EnemyManager.as, healthCrate.as, HighScores.as, Item.as, ItemManager.as, MyFavSong.as, pBulletSound.as, Player.as, ScoreBoard.as
2. Open CommandoStorm flash file
3. Compile CommandoStorm
4. Run CommandoStorm.swf file
5. Play!!

## Installation guide – Android

1. Download CommandoStorm.apk
2. Install ES File Explorer on your android device
3. Connect android device to computer – set to Media Transfer Device
4. Copy CommandoStorm.apk over to android device
5. Locate CommandoStorm.apk on android device using ES File Explorer
6. Click CommandoStorm.apk
7. Install -> Install -> Decline -> Open
8. Play!!

# Appendix 2

See zip folder for Commando\_Storm\_Storyboard

See zip folder for Commando\_Storm\_Presentation