University of Portsmouth

School of Creative Technologies

Final year Project undertaken in partial fulfilment of the requirements for the BSc (Honours) in **Computer Games Technology**

Effective use of Git Version Control for multi-platform game development

By

Alexander James Ryan Carter
778505

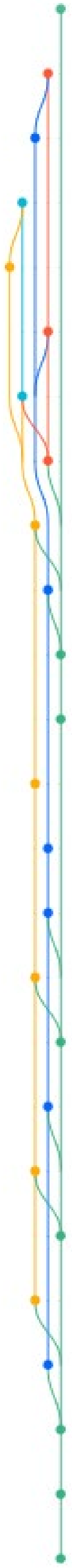
Supervisor: **Dr Peter Howell**
Project Unit: CT6CTPRO
May 2020

Project Type: **Combined**

Alexander James Ryan Carter
UP 778505

Abstract

This project investigates how Git Version Control can be used for Game Development for multi-platforms games. The research is conducted through a review of literature, findings from experimentation, and results of implementation into the development of a mobile game. The game produced is a dependency for the experiment to be conducted. The technology used in this project was Git, SourceTree (Visual Git client), Unity 2019 (Game Engine), Visual Studio (IDE), Adobe Photoshop (Pixel Manipulation), 3DSMax & Blender (3D Modelling), Google Drive (Online Documenting) and Microsoft Word (Word Processing). The findings demonstrate how when used in a specific workflow, Git can be used as a tool to aid developers in fluidly updating their games for each platform with minimal manual interaction. The experimentation demonstrates how a Unity project can have multiple versions of the same game on multiple platforms while remaining a single file structure on local developer machines as well as online backups. The project also outlines areas for further research and experimentation into potential alternative solutions.



Effective use of Git Version Control for multi-platform game development.

RESEARCH AND PROJECT DISSERTAION

ALEX CARTER – UP778505

Table of Contents

Terminology 4
1.0 Project Outline 5
2.0 Professional Development 6
3.0 Literature Review 6
3.1 What is Git? 6
3.2 Why Git? 7
3.3 Documentation 8
3.4 Challenges 8
4.0 Methodology 9
5.0 Prototyping 10
5.1 Setup 10
5.2 Initialisation 10
5.3 Device Specific Testing 11
5.4 Theme Testing 12
5.5 Findings 13
6.0 Implementation 15
7.0 Testing 16
8.0 Reflection 22
9.0 Unprecedented Encounters 23
10.0 Conclusion 24
11.0 Bibliography 25
12.0 Table of Figures 26
13.0 Table of Tables 26
14.0 Table of Appendices 26
Appendix 1 - Game Design Document 27
Appendix 2 - Blog Post 39

Terminology

As Version control and game development terminology is not widely known, provided is a description of terms found within this research project.

Multi-platform software is the ability for a piece of software to run on a variation of devices and operating systems (e.g. PC, Mac, Android, IOS etc.) while maintaining the same aesthetic and functionality.

Git is a piece of software which is the focus of this research project. It is what enables developers to create software in a team for an unlimited amount of variations.

A **Project** is any piece of software being created either by one or many developers. It will have a purpose and will have been designed to meet a selection of requirements.

Repositories (Repo's) are how Git stores all versions of a single project in one contained file structure. Files not 'checked out' are stored in a hidden 'git' folder so only files wanted are in the project at any one time.

Checked out is a term meaning which commit/version of the project you currently have on your local machine. These files will be tracked to see if any changes are made to them, and if so, allow you to keep and share, or remove and revert to their original state.

A **Remote** is an online hosting copy of the repository. This is what allows other developers to access the files from anywhere in the world, having an exact replica of the project as the rest of the team.

A **Commit** is a 'snapshot' of the changes to files by a developer. A repository is made up of commits by either one or many developers. Commits also require a description to outline what has been done since the last commit. This allows better tracking if something were to go wrong.

Branches are what allow developers to have their own copy of the project to work on without interfering with other developers. Developers can create their changes/new features here and merge to a shared branch when ready.

Merging is the act of combining branches. This will allow other developers to access the new changes/features once it is ready, rather than working on a project which has partially complete functionality.

A **Diff** is how developers can see what has been changed to a file before and after a commit by themselves or others.

An **IDE** is an Integrated Development Environment. This is a piece of software that allows developers to create software.

An **Enum** is a C# programming language feature which allows for a variable's value to be from a specific group of values. This is similar to a dropdown input field.

Unity is a game development engine. This software provides the tools a developer needs to create and build games for supported platforms.

1.0 Project Outline

This research will be around how to effectively use version control tools in order to simplify the development of a cross-platform game. Through literature review and experimentation, this research hopes to make it clearer on how to use version control efficiently for future game development.

The games industry is ever growing and with that, so does the number of platforms available for users to play them on. This becomes a challenge for developers as they need to make a build for each platform that they wish the game to be playable on. Even though game engines have been getting better at giving the options to build the same game to multiple platforms, it is not always as simple as switching the target and hitting build.

The issues arise when the technology or features you wish to implement is not cross-platform, meaning each platform you build for will differ slightly in the technology behind the scenes. An example of this is with the Google Play Games backend for leader boards, achievements and multiplayer networking available to android apps, but is not compatible with apps on the Apple IOS platform.

Unity (Unity Technologies, 2019) will be the game engine of choice in order to develop a simple game to conduct the experiment. For the version control technology, Git (Torvalds & Hamano, 2019) will be used alongside Bitbucket (https://bitbucket.com) as the Git remote and SourceTree (Atlassian, 2019) as the local visual interface into the Git repository.

The game itself is not part of the experiment but simply a means to conduct the research. The game will be developed with an additional designer/artist. The research and participant will have creative control on how the game is designed in order to implement the gameplay, features, technology and amending the scenes user interface to work across the various devices the game will be available on.

The version control repository will house both platform versions as well as a platform neutral version, being the main development of the base game without any technology backends implemented. The platform specific versions will only have the technology they require for their respective platforms.

By the end of the project there will be a duplicate of the same game, one using a differing backend technology to the other. To verify whether the use of version control has been effective, an update will be pushed to both versions and if no further necessary alterations must be made to those versions, then the experiment has been successful.

2.0 Professional Development

This research is important to me because I currently make mobile games for the Google Play Store, Amazons App Store and Apples App Store. This has caused issues when each platform has its own requirements and limitations where features or technology in one version is not Compatible with another. An example is when adding leader boards and achievements into games. Google have their Google Play Games Application Programming Interface (API) which works great for Android devices with the Google Play Services however, both Amazon and Apple devices do not allow this API to be used, forcing the developers to use the services offered by each corresponding platform.

Using different services for each platform also means player data is fragmented which creates barriers between competitive players wanting to compare their scores and achievements with their friends. As a solution to this specific issue, in the future I would like to experiment in the idea of creating my own online hosted games services offering cross-platform leader boards and achievements.

3.0 Literature Review

When first thinking about using version control, most wonder why they should use it. SoundSoftware states *"Version control reassures you and the people you collaborate with, gives you the confidence to carry out more ambitious experimental work, and makes your everyday working processes simpler and more satisfying."* (Soundsoftware, n.d.) which combined with Figure 1, demonstrates the strong opinions of the seemingly mandatory use of version control with any creative project from professionals in varying industries. The humour in the figure also represents the consensus from professionals in the industry, whereby if a team is not using Git, they are likely doing it wrong. *"Git is a very useful skill to have and almost necessary in many companies."* (Dev.to, 2020)

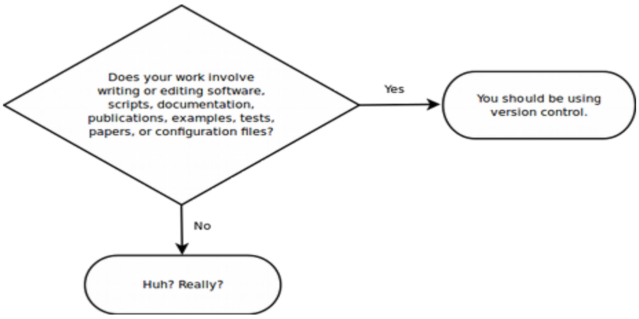

Does your work involve writing or editing software, scripts, documentation, publications, examples, tests, papers, or configuration files?
Yes
You should be using version control.
No
Huh? Really?

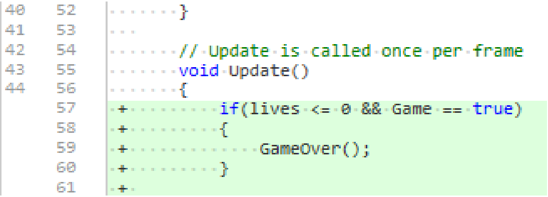
Figure 1 Should you use Version Control? (soundsoftware, nd)

3.1 What is Git?

Git was first released in 2005 and originally developed by Linus Torvalds. Torvalds created it in order to better manage the open-source nature of his other widely used software, the Linux Kernel. Linux used to be managed through a similar version control system known as BitKeeper (BitMover Inc., 2000), but decided to develop his very own alternative. There were some legal issues around the creation of Git competing with BitKeeper, however the main reasons for its existence was due to a new mandatory payed license agreement implemented in 2005. The slowness of the system was also evident when challenged with a project the size of the Linux Kernel, Torvalds wrote. *"Taking tens of seconds to apply a patch just because the source base is big is just not acceptable."* (McMillan, 2005).

The reasonable and sensible reason why Torvalds named his project 'Git' as still unknown however, he has given a somewhat comedic response to the question in an email interview with PCWorld, stating *"I'm an egotistical bastard, so I name all my projects after myself. First Linux, now git."* (McMillan, 2005)

Imagine a system that, rather than saving duplicate copies of your files every time it is changed, only the changes to a file is saved. This is Git in a nutshell. This helps keep the file size to a minimum and allows others to work on the same file with only their changes being added to file, rather than replace the entire file with their version. As shown in Figure 2, Git will 'track' the new changes to code with what was changed, the original line number and the new line number. Developers can see these changes to files in the 'Diff', which shows what is different from the last version.



```
40   52        }
41   53
42   54        // Update is called once per frame
43   55        void Update()
44   56        {
     57   +        if(lives <= 0 && Game == true)
     58   +        {
     59   +            GameOver();
     60   +        }
     61   +
```

Figure 2 Screenshot of changed code in SourceTree

3.2 Why Git?

When creating any type of creative project, creating versions is a good idea, whether it is for more than one platform or not. As Loeliger and McCullough state, *"No cautions, creative person starts a project nowadays without a back-up strategy. Because data is ephemeral and can be lost easily—through an errant code change or a catastrophic disk crash, say—it is wise to maintain a living archive of all work."* (Loeliger & McCullough, 2012).

Without Version control, ensuring a project goes smoothly is made much more difficult as you would have to save iterations of the project. This form of backing-up has three main restrictions. Firstly, being file size, as the amount of storage taken up will multiply by the amount of iterations, plus the additional amendments in each. The second restriction is if you need to roll back for any reason, you can only roll back to the last time you iterated. It will also be difficult to work out which saved copy is the one you want, without having larger than needed titles or supporting text documents. Lastly, unless you are also making additional backups offline or paying for large online storage, you could end up losing everything in the event of a disk corruption or if your development machine gets stolen.

For larger projects where more than one person is working on the same project, the lack of version control will be a hinder to the speed and efficiency of how they collaborate. This is another upside of Git where there is no limit to how many creative individuals can work on one project simultaneously.

Git has not got the best reputation for being user friendly nor easy to use as a beginner. There is a learning curve to be mastered. This, however, is the only drawback compared to the numerous benefits for a project. Git, at its core, is a command-based software. This means by default, there is no visual representation of the state of your project's repository. This is where much of the bad reputation comes from, as not everyone is comfortable with using command-based software and won't want to place their projects integrity on the press, or mis press, of blindly tapping a key. Therefore, visual interfaces have been created to give Git users a visual representation of what is happening. Git does have one built-in; however, it is only viewable after using a command within the corresponding repository. Due to the complexity this can have when working on more than one project, software like SourceTree were developed.

3.3 Documentation

As Zhang states, “A design doc is the most useful tool for making sure the right work gets done.” (Zhang, 2018). Therefore, the best place to start when creating a video game is the software design document. This allows a developer to break down the game into the smallest bite size details in order to start thinking about how it should be developed, with version control in mind.

The design document will include details about the technical details, requirements, game idea and mechanics. Further break down will describe the scripts needed to run the game and how they should be structured. This will allow the developer to have a better understanding of how the use of version control can aid in creating variations of the same script to work across the intended platforms.

3.4 Challenges

Joorabchi et al states *“Developers currently treat the mobile app for each platform separately and manually check that the functionality is preserved across multiple platforms.”* (Joorabchi, Meshbah, & Kruchten, 2013) which is still true today, however there are more tools to make this easier. One of the biggest issues when it comes to mobile development is fragmentation within the mobile market. Even though there are only a few popular platforms available (Android, IOS and Windows Phone) the fragmentation between operating system version, screen sizes, performance, technical restrictions and requirements etc make it difficult to develop for them all. *“Each mobile platform is different with regard to the user interface, user experience, Human Computer Interaction (HCI) standards, user expectations, user interaction metaphors, programming languages, API/SDK, and supported tools.”* (Joorabchi, Meshbah, & Kruchten, 2013)

Programming languages is an example of a technical hurdle between platforms. Android uses Java (Sun Microsystems, 1995) as its programming language for its operating system and apps, compared to Apple’s IOS which uses its own bespoke language, Swift (Apple Inc., 2014). Both Apple and Google have provided their own Integrated Development Environments (IDE) to allow developers to create apps on their respective platforms. The issue is because both use different languages, making an app for both can be tedious having to duplicate the work for both and knowing how to do so in the corresponding language. This is where Game Engines have made life easier for game developers specifically as many of the most popular engines have a built-in pipeline which runs the project through Googles and Apples compiler, which translates what you have done into their languages to run on devices. Such engines/builders don’t seem to be as widely available for apps which are not games. While this has reduced the workload for multi-platform development significantly, there are still hurdles to overcome.

Even with tools like game engines which support multi-platform, Perry et al found that the *“current tool, process, and project management support for this level of parallelism is inadequate…”* (Perry, Siy, & Votta, 2001). While this research was conducted in 2001, technological advances have been growing as the years progress which would mean the findings Perry et al found would likely be outdated. An example of this is how Unity are working on a new featured called ‘Distribution Portal’ which streamlines the publishing of games. From within the editor, you will be able to publish straight to the platforms store. (Unity Technologies , 2019)

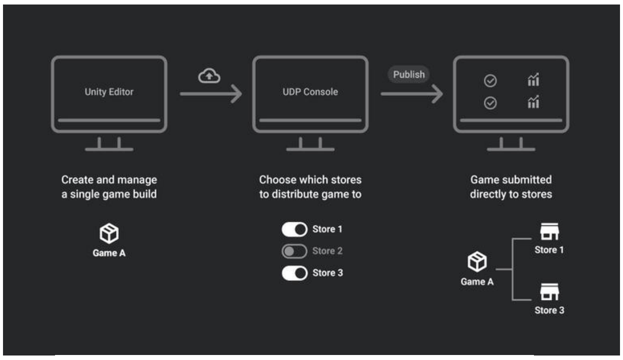

Unity Editor
UDP Console
Publish
Create and manage a single game build
Game A
Choose which stores to distribute game to
Store 1
Store 2
Store 3
Game submitted directly to stores
Game A
Store 1
Store 3

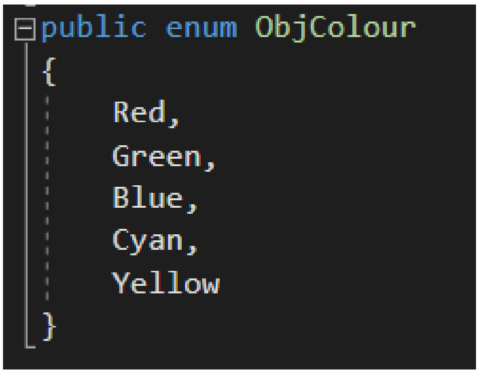
Figure 3 Unity's new Distribution Portal

4.0 Methodology

Much of the research will be conducted through experimentation and a review of existing literature. The literature chosen to support this study where from a variation of sources from professions in software and video game development. Focus points where chosen from different positions in a project lifecycle, e.g. from planning and writing a design document to how version control can aid in a multi-platform software development.

The design document is finalised as to what is intended to be created with milestone dates as to when certain parts of the development lifecycle should occur, e.g. alpha and beta builds. These dates can be seen in Appendix 1 - Game Design Document p11 along with a gantt chart of the production lifecycle. As design documents are working documents, there is some sections still to be filled in or yet to be determined (TBD).

Development of the artefact has been started and a build is already working on android of the base game. The technical back-ends for each platform will be added to start experimenting on how adding new features carry over to the device specific branches. Thought has gone into the coding of the game to make development easier with regards to the different intended platforms. An example of this is using Enums (programming variable type in C-Sharp) shown in Figure 4 and Appendix 1 - Game Design Document p9, which makes it easier to make variations of the objects in the game accessible through code much simpler than without. This also means that adding new objects to the game will be easier and little code will need to be added, meaning less changes to be merged to the device specific branches.



```
public enum ObjColour
{
    Red,
    Green,
    Blue,
    Cyan,
    Yellow
}
```

Figure 4 Screenshot of Enum variable type in C-Sharp

The experimentation to be conducted will be an important part in ensuring the project will run smoothly. This is important because it ensured that technology and features will work in the project before applying the tested approach onto the game in a pre-production state. Failure to experiment may result in delays to publication due to wasted time on implementation that where not fit for purpose. As shown in Figure 4, Experimentation will need to be conducted to confirm if the use of Enums has indeed made the workflow of adding new object types easier.

Throughout the project there will be frequent updates to the development studio's blog and social media which will be a point of feedback when the game is in alpha and beta testing. The first blog is available, shown in Appendix 2 - Blog Post. The blogs shall contain technical details in relation to the use of Git, Version control, game engine usage and programming for the game.

5.0 Prototyping

In order to conduct the research, a prototype experiment must be produced to receive immediate feedback and potential solutions to the problems identified. The prototype must be simple but have enough complexity to resemble creating various versions of the same game in addition to creating iterative updates after the fact. Key features the prototype must have are as follows:

- Folder for all files as a Git repository
- Simple text documents to resemble the games content
- Main development branch to resemble changes to base game
- Two or more branches to resemble platform specific versions
- An additional branch to resemble themed updates

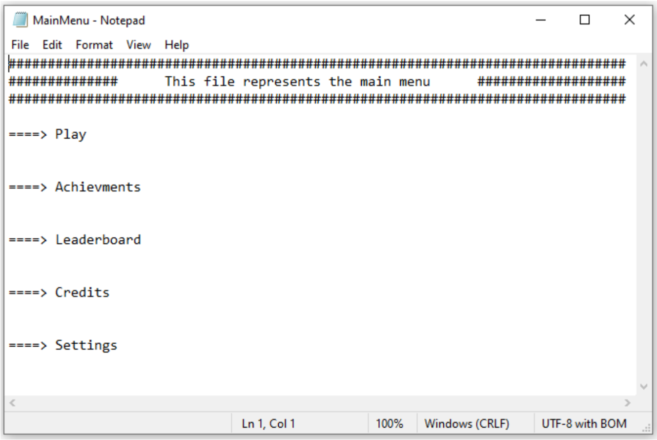
The Git repository should then also be connected to a Git remote to replicate hosting a version-controlled game with redundancy. This will allow the prototype to be available on any client development machine that has Git installed.

5.1 Setup

Setup of a version control repository starts with the remote, creating an online repository ready to be cloned to the client machine in preparation for development to begin. For this project, Bitbucket (https://bitbucket.com) was chosen as it allows unlimited private repositories with up to five users on their free tier. This is desired for small game developers so that their games' source code and Intellectual Property (IP) is protected from direct copyright, only allowing the public to see select published content. The other most common Git remote platform is GitHub (https://github.com/), however this is less desirable to smaller teams as their free tier limits the team to only public repositories, allowing anyone to access and download the source code of the game. For larger games, a paid plan or creating your own Git remote server would be necessary to overcome both Bitbucket and GitHub's storage restrictions. For transparency of the research, this prototype experiment will be made public. (https://bitbucket.org/Spectrolite-Studio/disso)

5.2 Initialisation

For the prototype to resemble a video game, files needed to be added to the repository. To keep things simple these files will be text (.txt) files. As seen in Figure 5, the first file created was named "MainMenu" which will be where changes can be added to resemble menu updates. Before committing this file to Git, a branch is created called "Base_Working" which will be the basis for the development of the 'game' before any theoretical device specific or theme amendments are made.



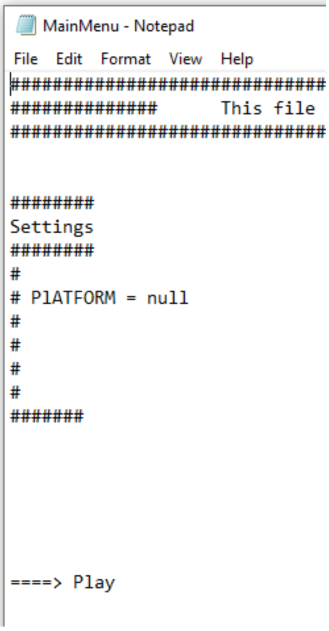
```
################################################################################
##############      This file represents the main menu      ####################
################################################################################

====> Play


====> Achievments


====> Leaderboard


====> Credits


====> Settings
```

Figure 5 Prototype Experiment 'MainMenu.txt' file

5.3 Device Specific Testing

Initial branches, following 'Base_Working' are the individual platform branches. These branches are created to hold the individual features and settings unique to each individual platform. These changes will only ever exist on these branches and branches that derive from it, but never back into 'Base_Working'. These branches were named 'Android_Working' and IOS_Working'.

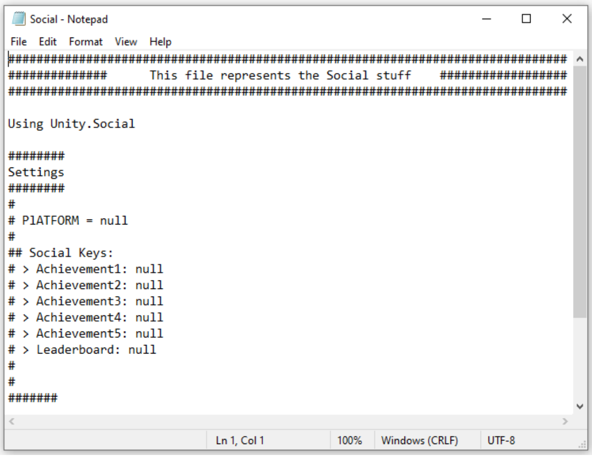
To keep track and to verify that device specific changes are staying on their respective branches, more detail was added to the files which states the platform the file should be for. On the Base branch the device was set to 'null', meaning nothing has been specified yet, as shown in Figure 6. This is then changed on the IOS and Android branches and should stay unchanged from then onwards. This platform specification section is applied to each file in this prototype to make it clear changes are being applied where planned.



```
##############################
##############      This file
##############################


########
Settings
########
#
# P1ATFORM = null
#
#
#
#
#######






====> Play
```

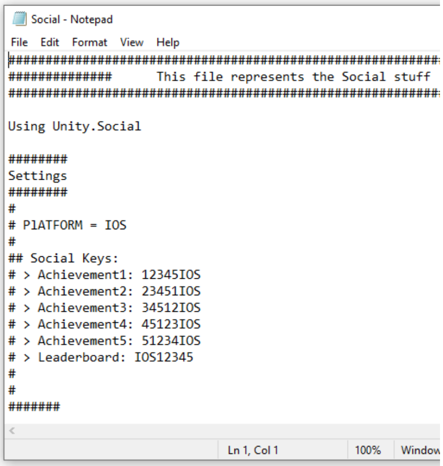
Figure 6 Prototype Experiment Menu file with platform set to 'null' on Base branch

To ensure prototype testing is as close to a real video game development as possible, a more rigorous and realistic implementation was added. As most mobile games have social features, e.g. leader boards and achievements, a file was created to keep track of theoretical 'API keys' which would be used as a reference to each leader board and achievement in a real game. These keys would also be device specific due to each platform having their own social services as outlined previously. This file was named 'Social.txt' and the 'keys' would be set to 'null' in the first instance on the Base branch until changed in the platform specific branches. This can be seen in Figure 7, with the platform specific versions shown in Figure 8 and Figure 9.



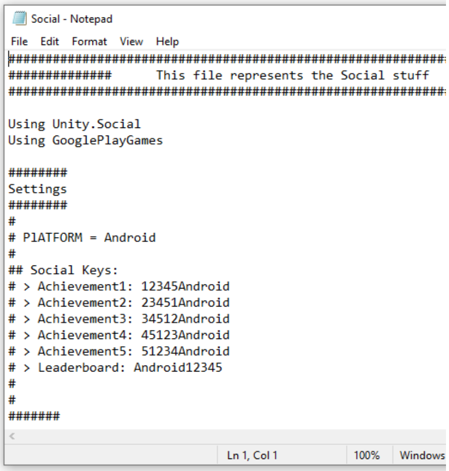
```
##############################################################################
##############      This file represents the Social stuff    #################
##############################################################################

Using Unity.Social

########
Settings
########
#
# P1ATFORM = null
#
## Social Keys:
# > Achievement1: null
# > Achievement2: null
# > Achievement3: null
# > Achievement4: null
# > Achievement5: null
# > Leaderboard: null
#
#
#######
```

Figure 7 Prototype Experiment 'Social.txt' file on Base branch



```
#############################################################
##############      This file represents the Social stuff
#############################################################

Using Unity.Social

########
Settings
########
#
# P1ATFORM = IOS
#
## Social Keys:
# > Achievement1: 12345IOS
# > Achievement2: 23451IOS
# > Achievement3: 34512IOS
# > Achievement4: 45123IOS
# > Achievement5: 51234IOS
# > Leaderboard: IOS12345
#
#
#######
```

Figure 8 Prototype Experiment 'Social.txt' file on IOS branch



```
##############################################################
##############      This file represents the Social stuff
##############################################################

Using Unity.Social
Using GooglePlayGames

########
Settings
########
#
# P1ATFORM = Android
#
## Social Keys:
# > Achievement1: 12345Android
# > Achievement2: 23451Android
# > Achievement3: 34512Android
# > Achievement4: 45123Android
# > Achievement5: 51234Android
# > Leaderboard: Android12345
#
#
#######
```

Figure 9 Prototype Experiment 'Social.txt' file on Android branch

5.4 Theme Testing

To simulate adding a theme to the game only using text documents, ASCII art was added. This is images made up of many characters from the ASCII (American Standard for Information Interchange) character table. The ASCII art chosen is shown below in Figure 10. which will be added to all text files to resemble changes that may have been made to introduce a theme to a game.

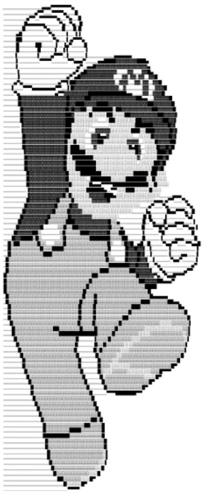
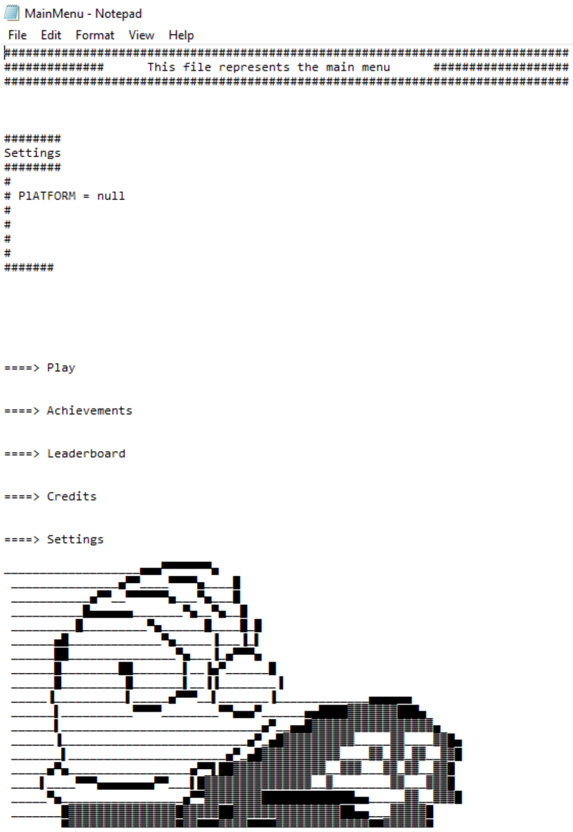


Figure 10 ASCII art of a video game character (Scratch, n.d.)



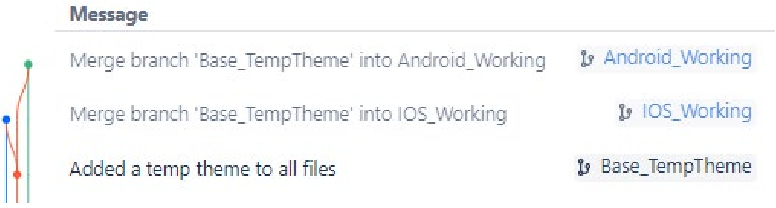
MainMenu - Notepad

```
###############################################################################
##############      This file represents the main menu      ###################
###############################################################################



########
Settings
########
#
# PlATFORM = null
#
#
#
#
#######




====> Play

====> Achievements

====> Leaderboard

====> Credits

====> Settings
```

Figure 11 MainMenu text file on the Base_TempTheme branch wit ASCII art

To start creating the themed version of the base game, a new branch was created which would contain the game along with the ASCII art, but no platform specific changes that exist in the already existing branches. This new branch was called “Base_TempTheme”. The ASCII art was then pasted at the bottom of each text file in the repository, as shown in Figure 11 with the ‘Platform’ setting remaining as ‘null’.

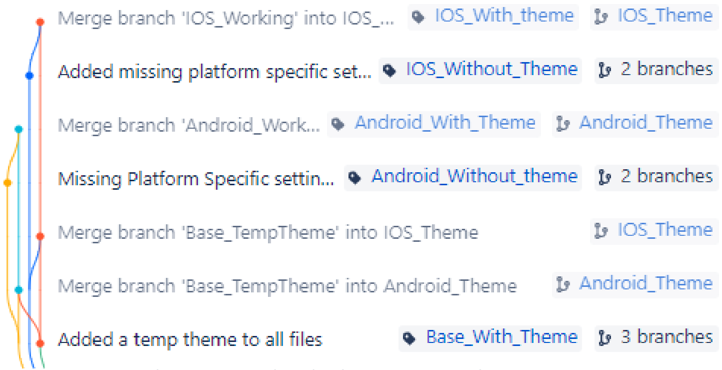
To test the theory that merging the theme into the platform specific branches directly would cause issues, the repository was forked, creating a replica of the project in its current condition. Within this forked repository, when merging the Base_TempTheme into Android_Working and IOS_Working, there where seemingly no issues or conflicts. This was expected, however there is now no efficient way for the theme to be easily removed from the Android and IOS branches, meaning a lot of extra steps would be involved, for every platform branch, in order to revert the changes and make a theme-less build. This is demonstrated in Figure 13 were the theme goes directly into the platform versions, leaving only two platform branches with variation appose to the desired four (two with the theme, and two without).



Message	
Merge branch 'Base_TempTheme' into Android_Working	Android_Working
Merge branch 'Base_TempTheme' into IOS_Working	IOS_Working
Added a temp theme to all files	Base_TempTheme

Figure 13 Forked repository for Theme -> Android & IOS mergers

This Git flow therefore does not work for efficiency nor redundancy. The only way to resolve this issue would be to then create a new device specific branch, e.g. Android_Working2, which would become confusing after many theme iterations as to which ‘working’ branch is still working.



Merge branch 'IOS_Working' into IOS_...	IOS_With_theme	IOS_Theme
Added missing platform specific set...	IOS_Without_Theme	2 branches
Merge branch 'Android_Work...	Android_With_Theme	Android_Theme
Missing Platform Specific settin...	Android_Without_theme	2 branches
Merge branch 'Base_TempTheme' into IOS_Theme		IOS_Theme
Merge branch 'Base_TempTheme' into Android_Theme		Android_Theme
Added a temp theme to all files	Base_With_Theme	3 branches

Figure 12 Theme -> Android_Theme & IOS_Theme mergers

Once returning to the original repository, the test was repeated but this time by creating a new branch from each platform ready for theme changes to be added. These branches where named “IOS_Theme” and “Android_Theme”. This then ensures there is a version of each platform with and without the theme in it, allowing a revert build to be made to the end users when the theme ends. These four versions are identified by tags in Figure 12.

5.5 Findings

The prototype testing undertaken was successful and demonstrated the potential efficiency of using Git for multi-platform game development if the correct Git-flow is followed strictly. The initial platform specific test show how easy Git allows for multiple versions of the same game can exist while working with the same source code with self-contained changes applied on top. These changes can be critical for a platform specific build to work correctly, for example a social API key. If these keys where overwritten from a base game update, this would not be noticeable in-engine and would then cause delays in development where unforeseen issues would occur and would have to be debugged to find the simple misconfiguration.

The theme testing worked well and provided clarity on potential issues with an opposing Git flow. In order to remain efficient and continue being a project with redundancy, there must always be a clean version of each platform specific game. This means it should work 'out of the box' whereby the project can be launched, built and work as expected for an end user. When adding a theme to the mix, this then adds an extra layer of potential complexity where there must be still a clean version of each platform, but also a clean theme version of each. The prototype testing demonstrated this working by at the end having a total of five versions of the same game; Base, Android, IOS, Android with theme and IOS with theme. In theory at any one time these versions should be able to be built ready for the end user.

In order to better visualise this Git flow, a diagram was made to show the order in which mergers should occur and to be followed strictly to keep each version clean and working.

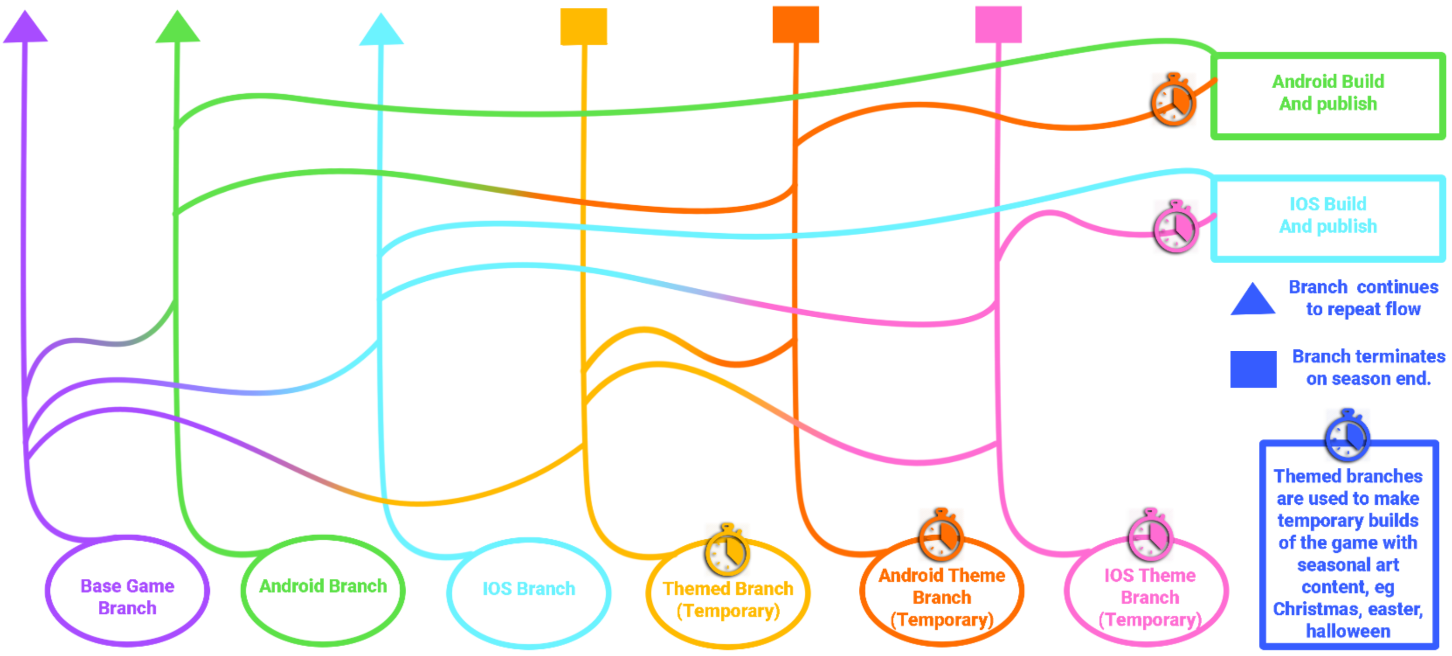

Android Build And publish
IOS Build And publish
Branch continues to repeat flow
Branch terminates on season end.
Themed branches are used to make temporary builds of the game with seasonal art content, eg Christmas, easter, halloween
Base Game Branch
Android Branch
IOS Branch
Themed Branch (Temporary)
Android Theme Branch (Temporary)
IOS Theme Branch (Temporary)

Figure 14 Git Findings Flow Graph.

Figure 14 shows the hierarchy of branches and in which direction the merges should happen. This is to ensure the Base Game contents stays clean of any settings which could conflict with content specific to each platform. The Base Game branch in this graph is the highest in the hierarchy, with device specific branches below, followed by the theme branch, then ending with the themed device specific branches. This flow shows how the further away from the Base Game the branches are, the more unique and specific the contents will be.

For the purpose of further explanation, see the below scenarios:

Scenario 1:
A game update is added onto the **Base branch.** For this change to be publicly available, Base must then be merged into the Android and IOS branches, built, and uploaded to their respected developer consoles.

- **Base -> Android -> Build and publish**
  - (merge changes on Base into Android)
- **Base -> IOS -> Build and publish**
  - (merge changes on Base into IOS)

Scenario 2:
It is November and a Christmas Update has been planned to be live for the duration of December. A Temporary branch is created off the Base branch to hold these temporary changes. Once the theme has been added, a temporary branch must be created for each platform to hold the device specific changes in addition to the theme. This is where the Android Theme and IOS Theme branches are created, off their respected platform branches. The **Temporary** branch is then merged into these new platform theme branches, adding the theme to each platform without interfering with the permanent device specific benches.

- **Base -> Temporary**
  - (new branch from Base with added theme changes)
- **Android -> Android Theme**
  - (new branch from Android)
- **IOS -> IOS Theme**
  - (new branch from IOS)
- **Temporary -> Android Theme -> Build and publish**
  - (merge theme into new android theme branch)
- **Temporary -> IOS Theme -> Build and publish**
  - (merge theme into new IOS theme branch)

Scenario 3:
It is the middle of December and a Christmas build is live on both the Apple and Google app stores. A bug has been fixed in the game which must go out to players immediately. This change has been applied to the Base branch and must now be merged into the Android and IOS branches. These branches do not contain the current Christmas theme, meaning more merges must be made to update the current live game. Base needs to be merged into the Temporary branch so we have a version of the game with the fix and the theme. This then needs to be merged into the Android Theme and IOS Theme branches. We can now build the theme branches to update the players versions, as well as have a build ready with the bug but without the Christmas theme ready for January.

- **Base -> Android**
  - (merge changes on Base into Android)
- **Base -> IOS**
  - (merge changes on Base into IOS)
- **Base -> Temporary**
  - (merge changes on Base into Temporary theme)
- **Temporary -> Android Theme -> Build and publish**
  - (merge temporary theme into android theme branch)
- **Temporary -> IOS Theme -> Build and publish**
  - (merge temporary theme into IOS theme branch)

6.0 Implementation

Due to circumstances outlined in section 9.0, the game in development will not be finished to publication standards as originally intended; however, will still meet the needs for testing the implementation of the Git flow.

The game was created using techniques aimed at making Git easier, for example using Prefab objects. Prefab's are Unity Game objects saved in a state and can allow changes to an Object in a scene without changing the scene file itself. The intention of this is to minimise the possibility of merge conflicts between branches with Scene files.

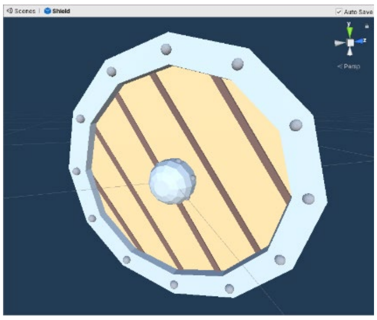


Figure 15 Game Object Prefab of a Shield

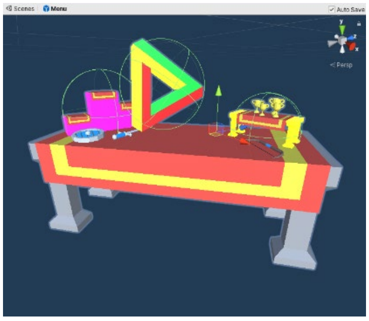
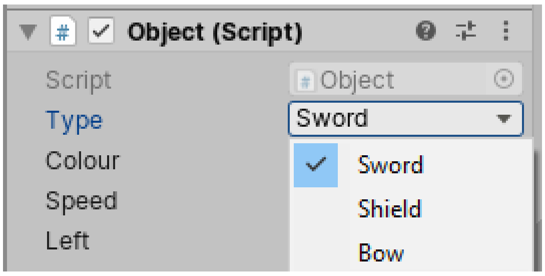


Figure 16 Game Object Prefab of the Menu Table

An example of a prefab implemented into the game was the objects being spawned. Each model to be spawned had a prefab made from it to ensure the scale was correct when spawned, as shown in Figure 15. This helped minimise the amount of code where scaling of objects is no longer needed to be done at run-time. Another prefab example is the menu table, shown in Figure 16. This table is used to show the player the Play, Leader board and Achievements icons which can be touched to load their functions. This table is a prefab as it means when adding a theme to the game, themed assets can be added to the table without affecting the scene file itself.

Another Technique used as outlined in in section 4.0 is using the C# programming language feature called Enums. This is a variable type which can define a state, where the value can only be one of a specified value, for example 'Sword' in Figure 17. Enums are used for the different type and colour of objects that are to be spawned. This should aid in reducing the amount of programming needed when changing to a theme as only the object type's will be needed to change. Another C# feature was used alongside the Enums which was a Switch Case. This is an alternate to using 'If statements' where the value of each Enums has a contained section of code to be called. This implementation can be seen in Figure 18 where a Switch Case is used to spawn the correct prefab object, depending on the value of the Enum type.



Object (Script)

Script: Object

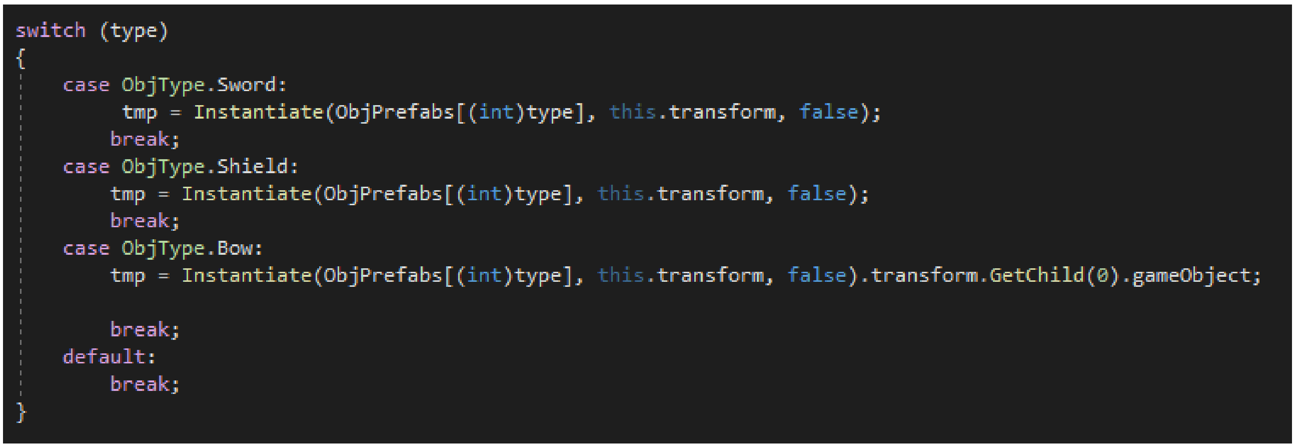
Type: Sword

Colour

Speed

Left

Figure 17 Enum Dropdown in Unity Editor Inspector.



```
switch (type)
{
    case ObjType.Sword:
         tmp = Instantiate(ObjPrefabs[(int)type], this.transform, false);
        break;
    case ObjType.Shield:
        tmp = Instantiate(ObjPrefabs[(int)type], this.transform, false);
        break;
    case ObjType.Bow:
        tmp = Instantiate(ObjPrefabs[(int)type], this.transform, false).transform.GetChild(0).gameObject;

        break;
    default:
        break;
}
```

Figure 18 C# Switch Case for spawning the correct prefab from Enum type

7.0 Testing

Throughout testing, screenshots were taken from the editor and device. This is to keep a record of how the game looks before and after merging into the device specific branches. Any discrepancies in visual or usability will highlight if testing was successful or not.

Table 1 Base -> Android

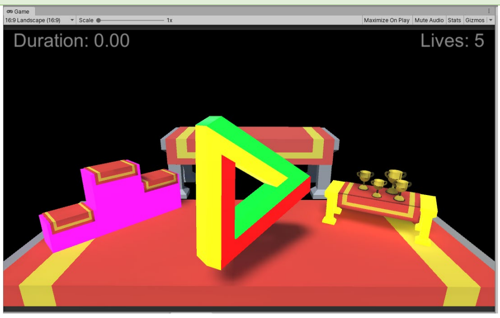
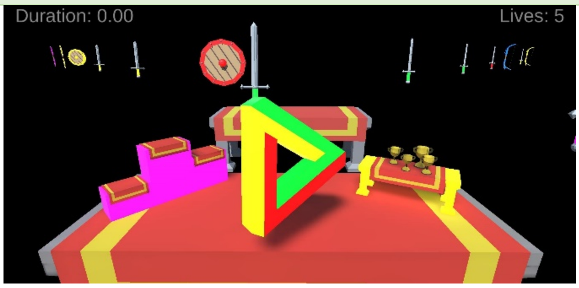
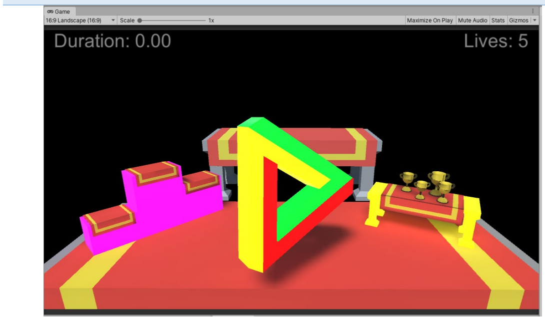
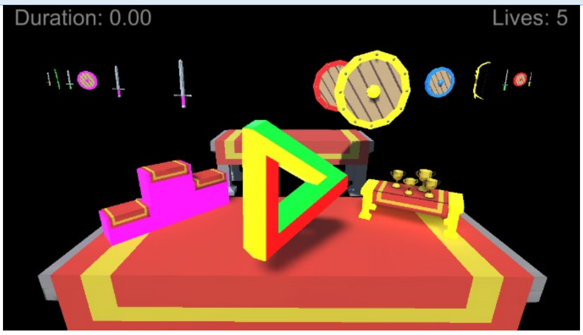
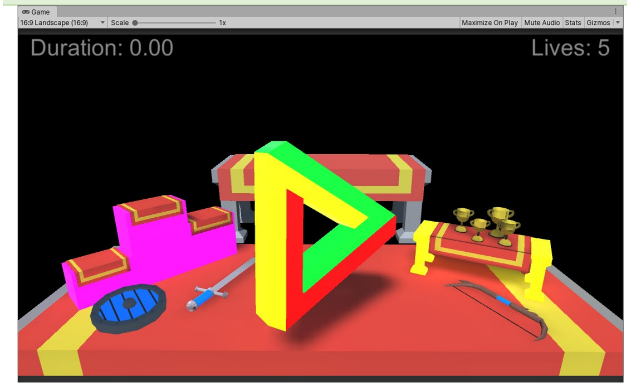
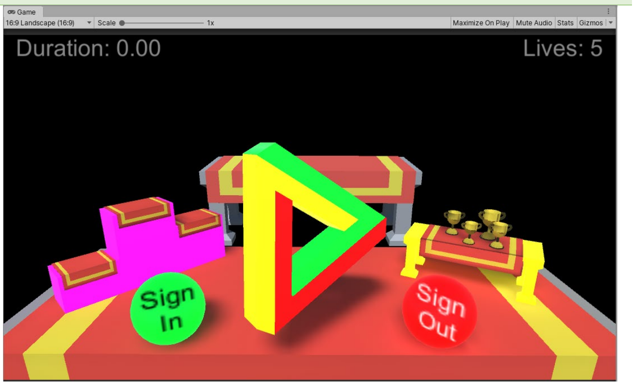
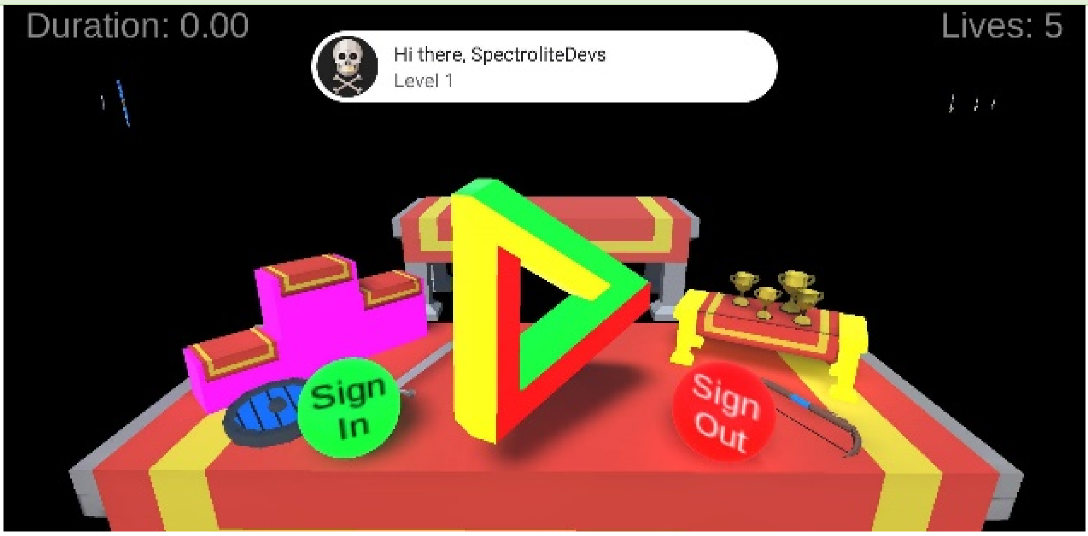
Base branch		**Android branch**
 **Base branch Editor screenshot**	+	**Android Build Settings**
	=	
 **Android build screenshot**		

Table 2 Base -> IOS

Base branch		Device screenshot
 **Base branch Editor screenshot**	+	**IOS Build Settings**
	=	
 **IOS build Screenshot**		

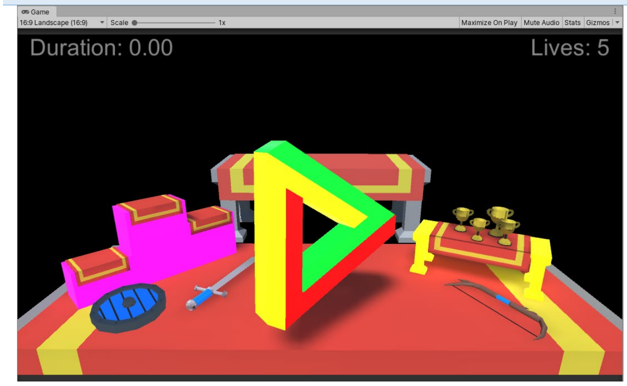
The outcome of the first series of test shown in Table 1 and Table 2 were successful while also expected. There were no device specific changes other than those required to build the game to the device. The next step was to add social features in the form of Google Play Games and Game Centre for Android and IOS to their specific branches. A change will then be made to the game and merged into those branches to simulate a game update. To succeed this test, no further changes should be required to the game between merging and building.

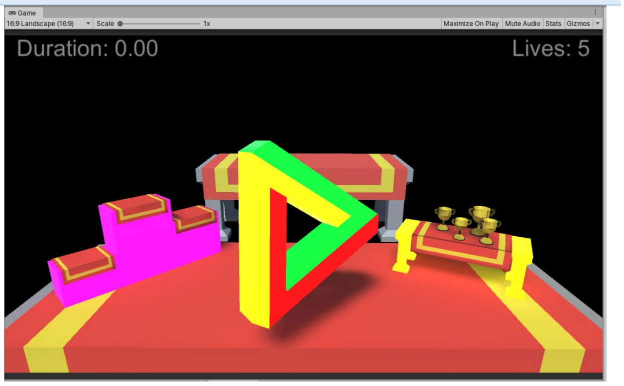
Table 3 Android -> Android + Social

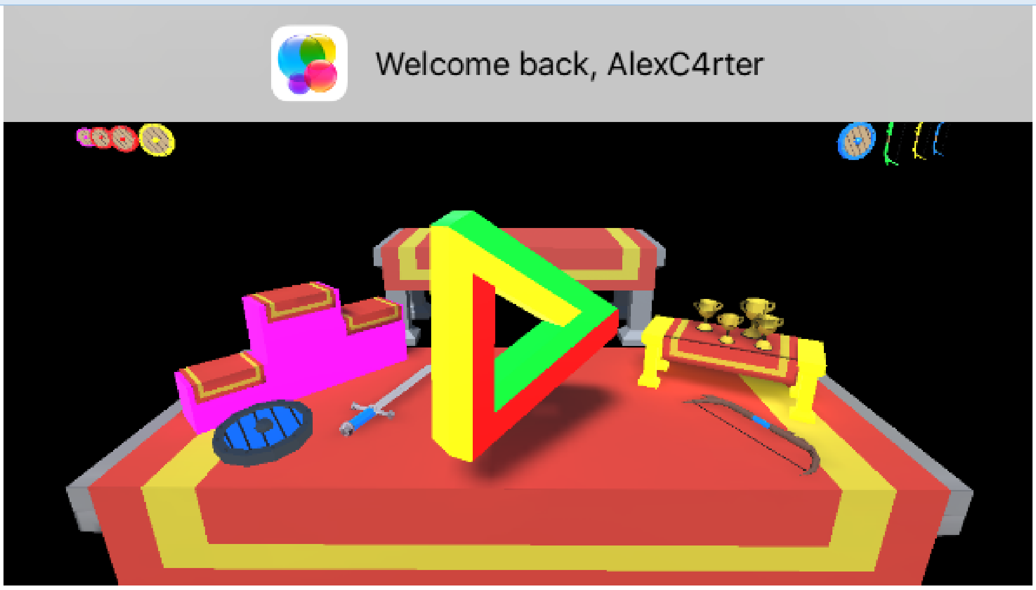
Base branch		**Android branch**
 **Base branch Editor screenshot with added items to table prefab**	+	 **Android branch with social features and Sign in/out buttons added to scene.**
	=	
 Duration: 0.00 Hi there, SpectroliteDevs Level 1 Lives: 5 Sign In Sign Out **Android build with social**		

After adding the Google Play Games social features onto the Android specific branch, merging an update from the Base branch and building the game worked with no erros. This merge demenstrated in Table 3 shows how changes that exist in the Android branch (the sign in/out buttons) carry over as well as the additions in the Base branch (the cosmetic items). Google requires games using Google Play Games to have the option to sign in and out. This is different for IOS with Game Center where Apple does not allow users to change their Game Center account from within individual apps.

Table 4 Base -> IOS + Social

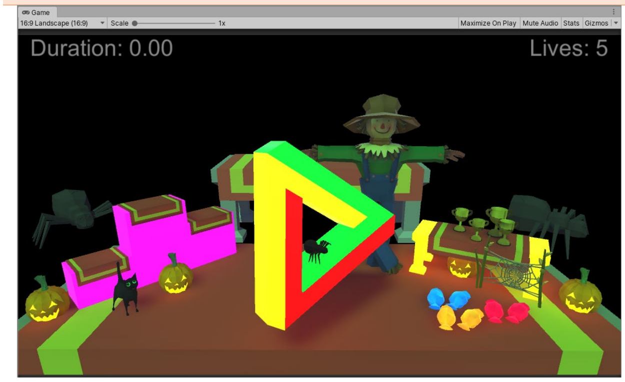
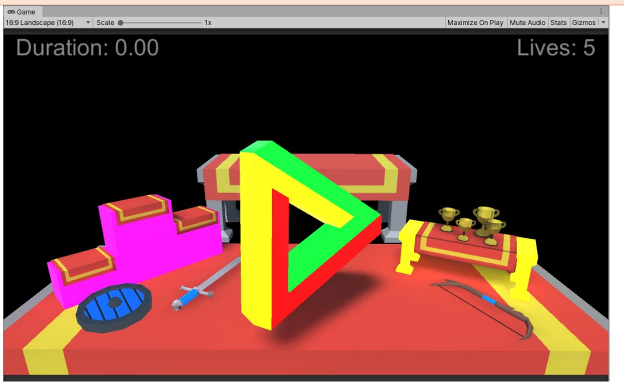
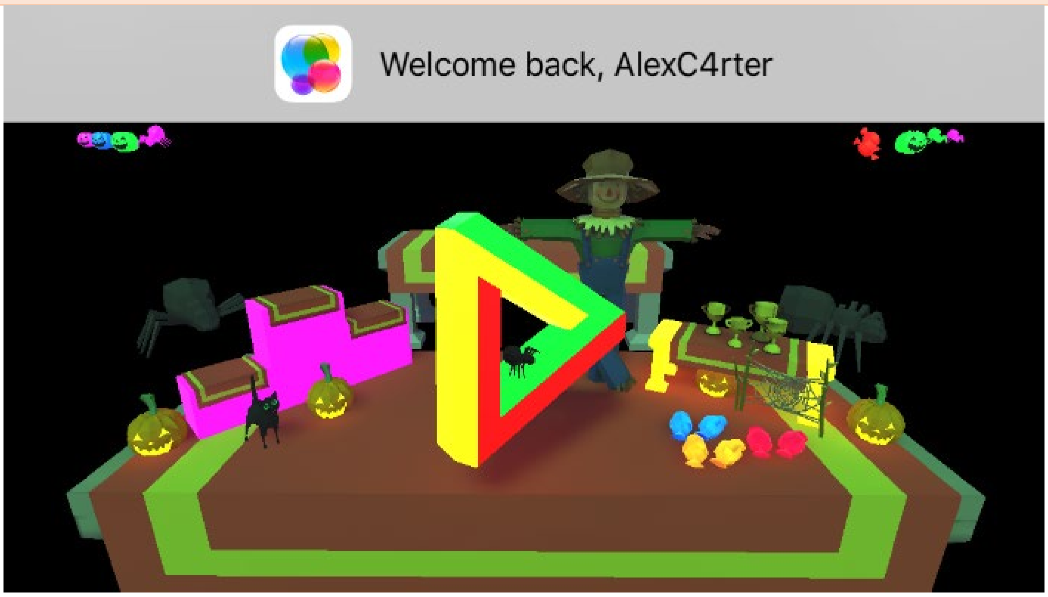
<table>
<tr><th>Base branch</th><th></th><th>IOS branch</th></tr>
<tr><td>

Base branch Editor screenshot with added items to table prefab</td><td>+</td><td>

IOS branch with social features. No extra visual changes. Game Center added to social buttons.</td></tr>
<tr><td colspan="3">=</td></tr>
<tr><td colspan="3">
Welcome back, AlexC4rter
IOS build with social</td></tr>
</table>

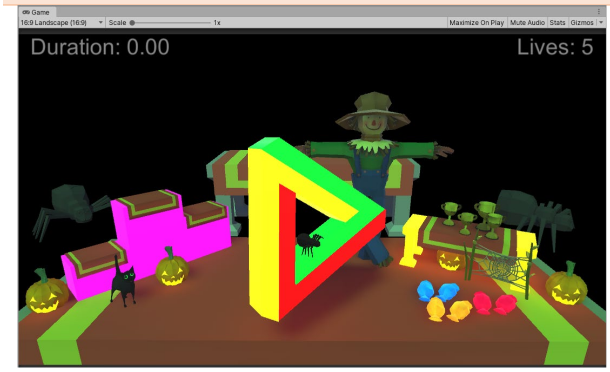
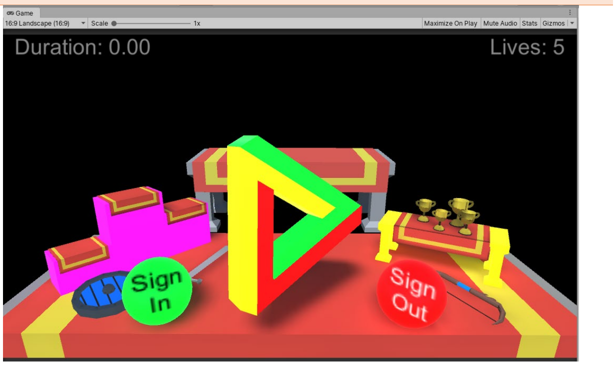
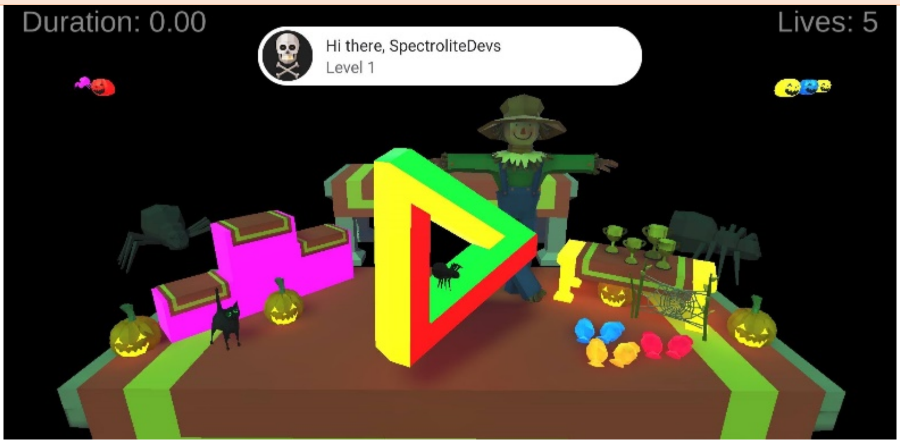
The Base -> IOS merge was successful, as seen in the resulted screenshot in Table 4 from the IOS Device with Game Center activated. As the IOS version does not contain any visual additions not already in the Base branch, the merge was not as challenging compared to the Android merge. This merge demonstrates how differences in the social features stayed in the IOS branch after merge.

Table 5 Theme -> IOS Theme Branch

<table>
<tr><th>Theme branch</th><th></th><th>IOS branch</th></tr>
<tr><td> Duration: 0.00
Lives: 5
Theme branch Editor screenshot with added items to table prefab and lighting in scene</td><td>+</td><td> Duration: 0.00
Lives: 5
IOS branch Editor screenshot with social features</td></tr>
<tr><td colspan="3">=</td></tr>
<tr><td colspan="3">Welcome back, AlexC4rter
IOS social + theme</td></tr>
</table>

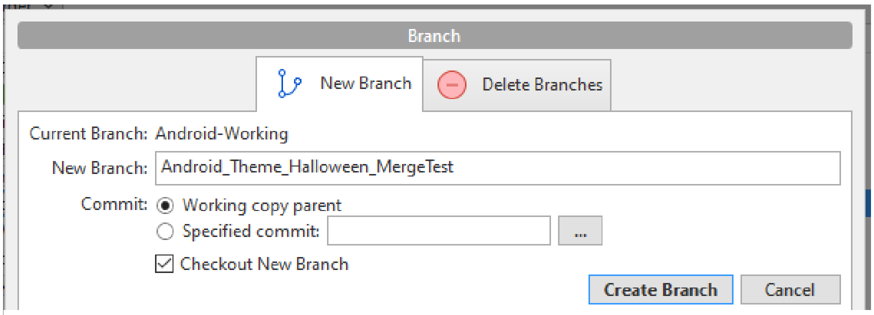
Adding a theme to the game was the most challenging task for Git with the amount of changes having to be made in one go. This was not as difficult for the IOS Themed version as there are no visual differences between the Base and the IOS versions, as shown in Table 5. This merge was a success and shows how the cosmetic items in the Base game where replaced with the items in the theme branch.

Table 6 Theme -> Android Theme Branch

Theme branch		Android branch
	+	
Theme branch Editor screenshot with added items to table prefab and lighting in scene		**Android branch Editor screenshot with social features**
	=	
	 Duration: 0.00 Lives: 5 Hi there, SpectroliteDevs Level 1 **Android social + theme**	

The Android Themed Version merge did not pass the initial test. This was due to a merge conflict, resulting in a choice of keeping the theme changes to the scene, or the sign in/out social buttons in the scene. As shown in the result device screenshot in Table 6, the android build now has the Halloween theme with social features but have lost the ability to sign in/out manually. Ways around this would be to make the sign in/out buttons a prefab spawned from a script at run-time; however, is not desirable.

After some further research into the issue, a tool built into Unity for this task was discovered. This Tool is called UnityYAMLMerge and is a command-line tool accessible to Git clients. This tool's purpose is to *"…merge scene and prefab files in a semantically correct way."* (Unity Technologies , 2019) which cannot be handled cleanly with conventional Git tools alone. After integrating this tool into the Sourcetree Git client, the merge was tested again on a new branch as demonstrated in Figure 19 below.



Current Branch: Android-Working

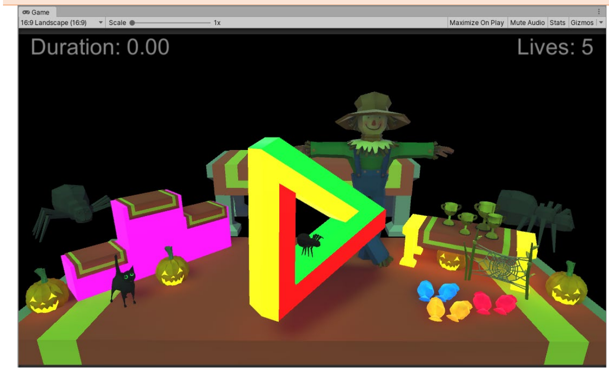
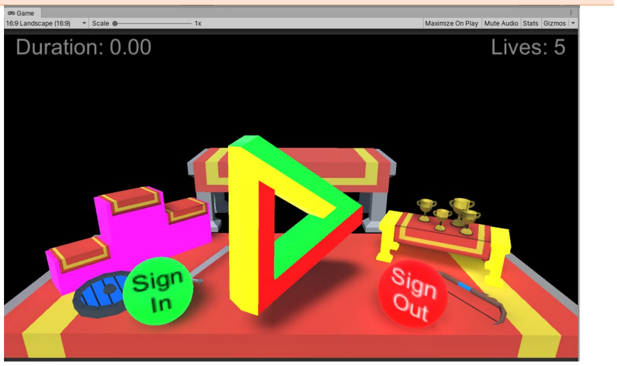
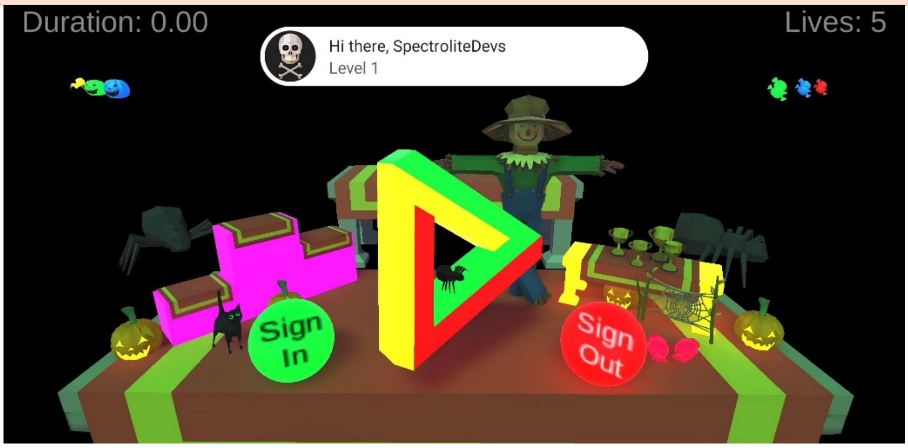
New Branch: Android_Theme_Halloween_MergeTest

Commit: ● Working copy parent
○ Specified commit:

☑ Checkout New Branch

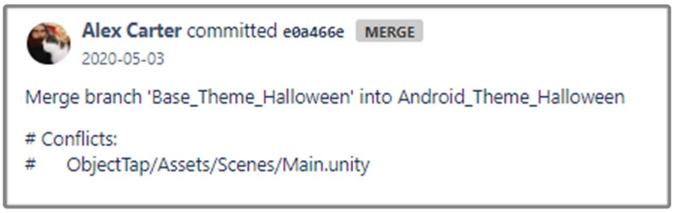
Figure 19 Git test branch for the UnityYAMLMerge tool.

Table 7 Theme -> Android Theme Branch (Using UnityYAMLMerge)

Theme branch		**Android branch**
 **Theme branch Editor screenshot with added items to table prefab and lighting in scene**	+	 **Android branch Editor screenshot with social features**
	=	
	 Duration: 0.00 Lives: 5 Hi there, SpectroliteDevs Level 1 Sign In Sign Out **Android social + theme**	

When repeating the same merge once again, the merge conflict appeared but this time allowed for the external tool to be used to handle the conflict. Once this tool had completed its task, a build was made and succeeded. The new Android device screenshot in Table 7 shows both scenes' changes now successfully exist in the Android theme branch.

As seen in Figure 20, The merge was successful, and an extra note was added to the commit message informing of the conflict and specifying which file in particular was involved. This allows for better tracking if something where to have gone wrong in the project and can be pointed back to this specific diff resolution. As the merge was successful, no further actions were required; however, if an issue persisted to exist then further research into alternative Unity diff tools would be required, or manual intervention to correct the conflict within Unity.



Alex Carter committed e0a466e MERGE
2020-05-03

Merge branch 'Base_Theme_Halloween' into Android_Theme_Halloween

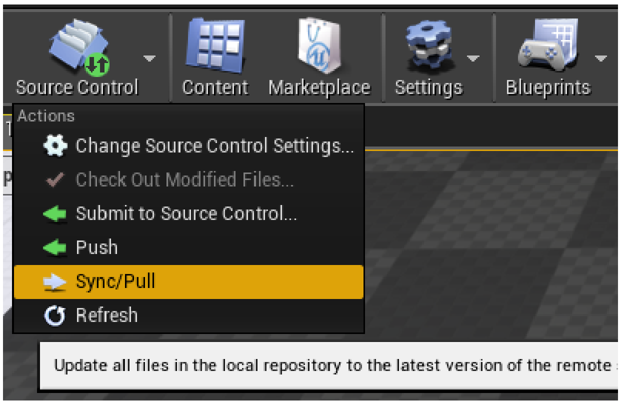
```
# Conflicts:
#	ObjectTap/Assets/Scenes/Main.unity
```

Figure 20 Successful diff resolution from merge conflict

8.0 Reflection

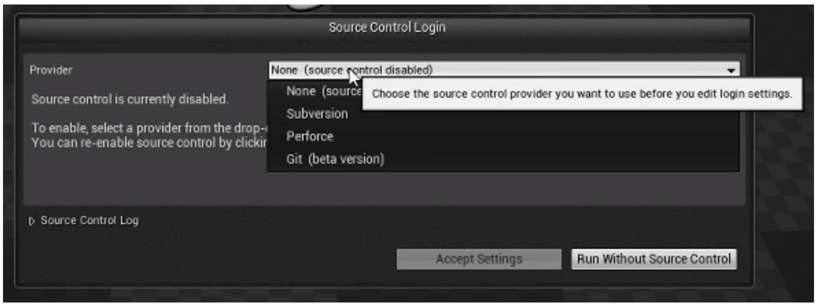
The test results produced from both the prototype and implementation reflect how Git can be used for a multi-platform game development on a small to medium scale. The tests where exhaustive to a familiar level of complexity found within the development teams' games and sets a good example for a future implementation of the Git flow identified in section 5.5. The experimentation demonstrated how Git has the ability to be used as a game development tool alongside having the expected features such as redundancy, online backup, collaboration, feature testing and project version handling.

The findings of the UnityYAMLMerge tool was great as it appears to work without issue for Unity Scene files, one of the concerns causing for this research to be conducted. While the tool only had to be used once, it set a good expectation of its potential for future merge conflicts. The tool is relatively hidden within the Unity Engine's source files and its presence in-engine is seemingly non-existent when using Git. It seems when it comes to version control in game engines, Unity still has progress to be made in order to make it a more seamless integration and reflect its importance within the game development industry. Unreal Engine 4 (Epic Games, 2014) has supported in-editor access to version control tools since the launch of version 4. This can be seen in Figure 21 showing the 'Source Control' options for an unreal project.


Source Control
Content
Marketplace
Settings
Blueprints
Actions
Change Source Control Settings...
Check Out Modified Files...
Submit to Source Control...
Push
Sync/Pull
Refresh

Update all files in the local repository to the latest version of the remote

Figure 21 Unreal Engine 4's built-in version control

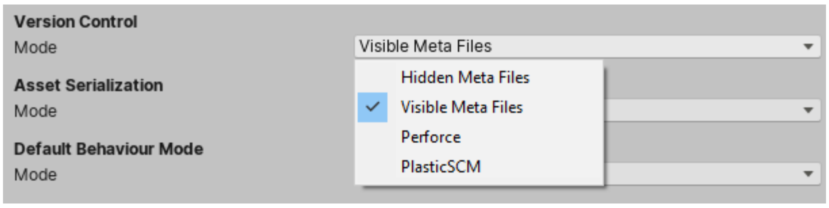


Source control is currently disabled.

To enable, select a provider from the drop-
You can re-enable source control by clickir

Choose the source control provider you want to use before you edit login settings.

Figure 22 Unreal Engine 4 Version Control Support options



Version Control
Mode: Visible Meta Files

Asset Serialization
Mode

Default Behaviour Mode
Mode

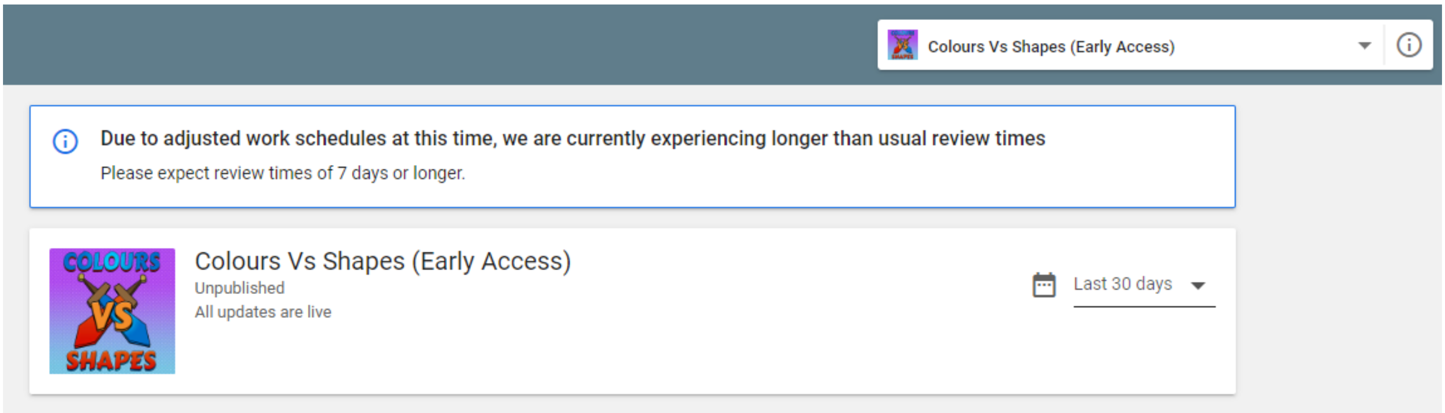
Figure 23 Unity 2019 Version Control Support options

As demonstrated in Figure 22, Unreal supports Git, Subversion and Perforce version control solutions out of the box, whereas Unity only supports Perforce and PlasticSCM, shown in Figure 23. Unity does however have the option to make Meta files visible and serialise binary files into a text format, so files are more easily read and understood by other solutions, like Git. If Unity had this level of integration in-editor for Git it would widen the possibilities to view diff's in the Editor, seeing exactly which changes may be conflicting and have greater control around how to handle a merge.

9.0 Unprecedented Encounters

As of January 2020, the world saw one of the largest and trickiest threats in recent history with the spread of the newly discovered Covid-19 virus, developing into a worldwide pandemic by March 2020. This Has affected everyone on every scale be it mourning loss, job losses, staff shortages, financial implications, mental health, logistical challenges, production vs demand, pressure on the internet infrastructure, challenges of working from home, and the knock-on effect for the entire education system. This project was no exception from becoming affected by this. Here I will outline what issues were encountered.

The original plan for this project was to, by the end, have a published game on both the Google Play Store and the Apple App Store. This was to ensure the research was accurate to the purpose of a real, published game in development. This would allow for the public to provide feedback on any issues and the general user experience for each platform on a handful of each sub-device. The official release of the game was pushed back due to staff shortages at Google, as evident in Figure 24 below. This staff shortage meant that every app update uploaded would encounter a review time of longer than a week before it could be even internally tested. This added a delay to each iterative update which meant that the game could not be tested by the development team as quickly, nor be tested by external testers for feedback. The same review times where implemented into Apple's App Store which meant also without a Mac computer, even with the source code, it is impossible to make a build to test with the development team at all.



Due to adjusted work schedules at this time, we are currently experiencing longer than usual review times
Please expect review times of 7 days or longer.

Colours Vs Shapes (Early Access)
Unpublished
All updates are live

Figure 24 Google Play Console longer review times

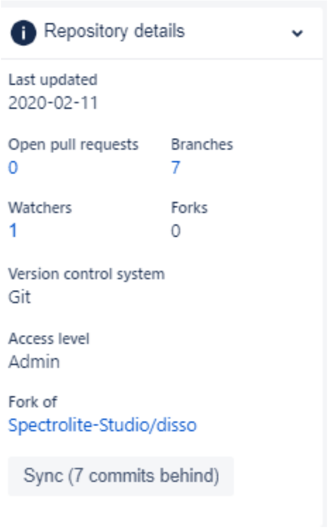
Due to the decision that the game should not be published, the blog posting of development progress was also ceased. This meant that the progress of the game could not be publicly documented and reduce confusion around its development and release. A later release date is planned for, once development and testing can continue as normal.

10.0 Conclusion

In conclusion to the research and project testing undertaken, an efficient Git flow has been discovered which works well for the scenarios outlined in section 5.5. This Git flow has succeeded in testing the prototype and implementation for reducing the number of extra steps required to update a game on more than one platform. While an extra step was required during the implementation due to a merge conflict, this still resulted in no manual repairs to the game's source files itself for continuous functionality to proceed and work as intended.

For the time being, Git and the Git flow outlined is a capable solution for multi-platform game development. While alternative solutions serve well for larger teams and organisations, more research and testing would need to be conducted for the specific scenarios defined. Perforce is a popular choice for large game development studios, as evident as being the main supported version control provider in both the Unity and Unreal engines. While Perforce is free for small teams, it does require the remote server to be self-hosting either on premises or in the cloud, introducing more running costs than the current implementation of Bitbucket or GitHub.

After using the 'Fork' feature in Git during the prototype test, it appears this could also be an alternative Git flow option. In this flow, a Fork of the project would be made, creating a replica in its current state. This Forked version could then be used to contain a themed version, removing the possibility of human error when it comes to merging the theme into the device specific branches. If a bug fix or an update is needed for the base game, this could then be applied to the original repository and synced to the forked clone, applying the latest commits on the existing branches. This option to sync changes between remote repositories is shown in Figure 25. In this Git flow, only the Forked repositories would contain theme branches. This Git flow would require more testing as a possible alternative to the current solution in order to compare the pros and cons of each method.



Repository details

Last updated
2020-02-11

Open pull requests
0

Branches
7

Watchers
1

Forks
0

Version control system
Git

Access level
Admin

Fork of
Spectrolite-Studio/disso

Sync (7 commits behind)

Figure 25 Repository description from Forked repo, allowing to Sync the most recent changes from the original

Considering the support both Unity and Unreal 4 have for Perforce, it seems this would also be an area for further research and experimentation in order to compare the other solutions to version control used in the industry. While this may incur additional costs, it could be a better solution for the scaling of projects and teams. This experimentation would have to be done is a similar manner to the research covered in this project where having multiple branches of the project for each platform is a requirement. Due to limited research and documentation on this specific topic it would be beneficial to compare the advantages and possible drawbacks.

11.0 Bibliography

Apple Inc. (2014). Swift [Programming Language]. California : Apple Inc.

Atlassian. (2019). Sourcetree . *(Version 3.2.1) [Computer software]*. Atlassian.

BitMover Inc. (2000). BitKeeper. *[Computer Software]*.

Dev.to. (2020, Jan 8). *Git cheatsheet for beginners*. Retrieved from DEV: https://dev.to/duomly/git-cheatsheet-for-beginners-5apl

Epic Games. (2014). Unreal Engine. *(Version 4.1) [Computer Software]*. Epic Games.

Joorabchi, M. E., Meshbah, A., & Kruchten, P. (2013). *Real Challenges in Mobile App Development.* New Jersey: Institute of Electrical and Electronics Engineers.

Loeliger, J., & McCullough, M. (2012). *Version Control with Git (2nd ed.).* California: O'Reilly Media Inc.

McMillan, R. (2005, April 20). *After controversy, Torvalds begins work on "git"*. Retrieved from PCWorld: https://www.pcworld.idg.com.au/article/129776/after_controversy_torvalds_begins_work_git_/

Perry, D. E., Siy, H. P., & Votta, L. G. (2001). *Parallel changes in large-scale software development: an observational case study.* Texas: ACM Transactions on Software Engineering and Methodology.

Scratch. (n.d.). *scratch.mit.edu*. Retrieved from Text Art!: https://scratch.mit.edu/discuss/m/topic/280694/

Soundsoftware. (n.d.). *Soundsoftware.ac.uk*. Retrieved from Why use version control?: http://soundsoftware.ac.uk/why-version-control

Sun Microsystems. (1995). Java [Programming Language]. California : Oracle.

Torvalds, L., & Hamano, J. (2019). Git. *(version 2.21) [Computer software]*. Retrieved from https://git-scm.com/

Unity Technologies . (2019). *Smart Merge*. Retrieved from Unity Documentation: https://docs.unity3d.com/Manual/SmartMerge.html

Unity Technologies . (2019). *Unity Distribution Portal (Beta)*. Retrieved from Unity.com: https://unity.com/unity-distribution-portal

Unity Technologies. (2019). Unity. *(Version 2019.3) [Computer software]*. Unity Technologies.

Zhang, A. (2018, July 13). *How to write a good software design doc*. Retrieved from freecodecamp.org: https://www.freecodecamp.org/news/how-to-write-a-good-software-design-document-66fcf019569c/

12.0 Table of Figures

Figure 1 Should you use Version Control? (soundsoftware, nd) 6
Figure 2 Screenshot of changed code in SourceTree........ 7
Figure 3 Unity's new Distribution Portal........ 8
Figure 4 Screenshot of Enum variable type in C-Sharp........ 9
Figure 5 Prototype Experiment 'MainMenu.txt' file........ 10
Figure 6 Prototype Experiment Menu file with platform set to 'null' on Base branch........ 11
Figure 7 Prototype Experiment 'Social.txt' file on Base branch........ 11
Figure 8 Prototype Experiment 'Social.txt' file on IOS branch........ 11
Figure 9 Prototype Experiment 'Social.txt' file on Android branch 11
Figure 10 ASCII art of a video game character (Scratch, n.d.) 12
Figure 11 MainMenu text file on the Base_TempTheme branch wit ASCII art 12
Figure 12 Theme -> Android_Theme & IOS_Theme mergers........ 12
Figure 13 Forked repository for Theme -> Android & IOS mergers........ 12
Figure 14 Git Findings Flow Graph. 13
Figure 15 Game Object Prefab of a Shield 15
Figure 16 Game Object Prefab of the Menu Table 15
Figure 17 Enum Dropdown in Unity Editor Inspector. 15
Figure 18 C# Switch Case for spawning the correct prefab from Enum type 15
Figure 19 Git test branch for the UnityYAMLMerge tool........ 20
Figure 20 Successful diff resolution from merge conflict 21
Figure 21 Unreal Engine 4's built-in version control........ 22
Figure 22 Unreal Engine 4 Version Control Support options........ 22
Figure 23 Unity 2019 Version Control Support options........ 22
Figure 24 Google Play Console longer review times........ 23
Figure 25 Repository description from Forked repo, allowing to Sync the most recent changes from the original 24

13.0 Table of Tables

Table 1 Base -> Android 16
Table 2 Base -> IOS........ 16
Table 3 Android -> Android + Social........ 17
Table 4 Base -> IOS + Social........ 18
Table 5 Theme -> IOS Theme Branch........ 19
Table 6 Theme -> Android Theme Branch 20
Table 7 Theme -> Android Theme Branch (Using UnityYAMLMerge) 21

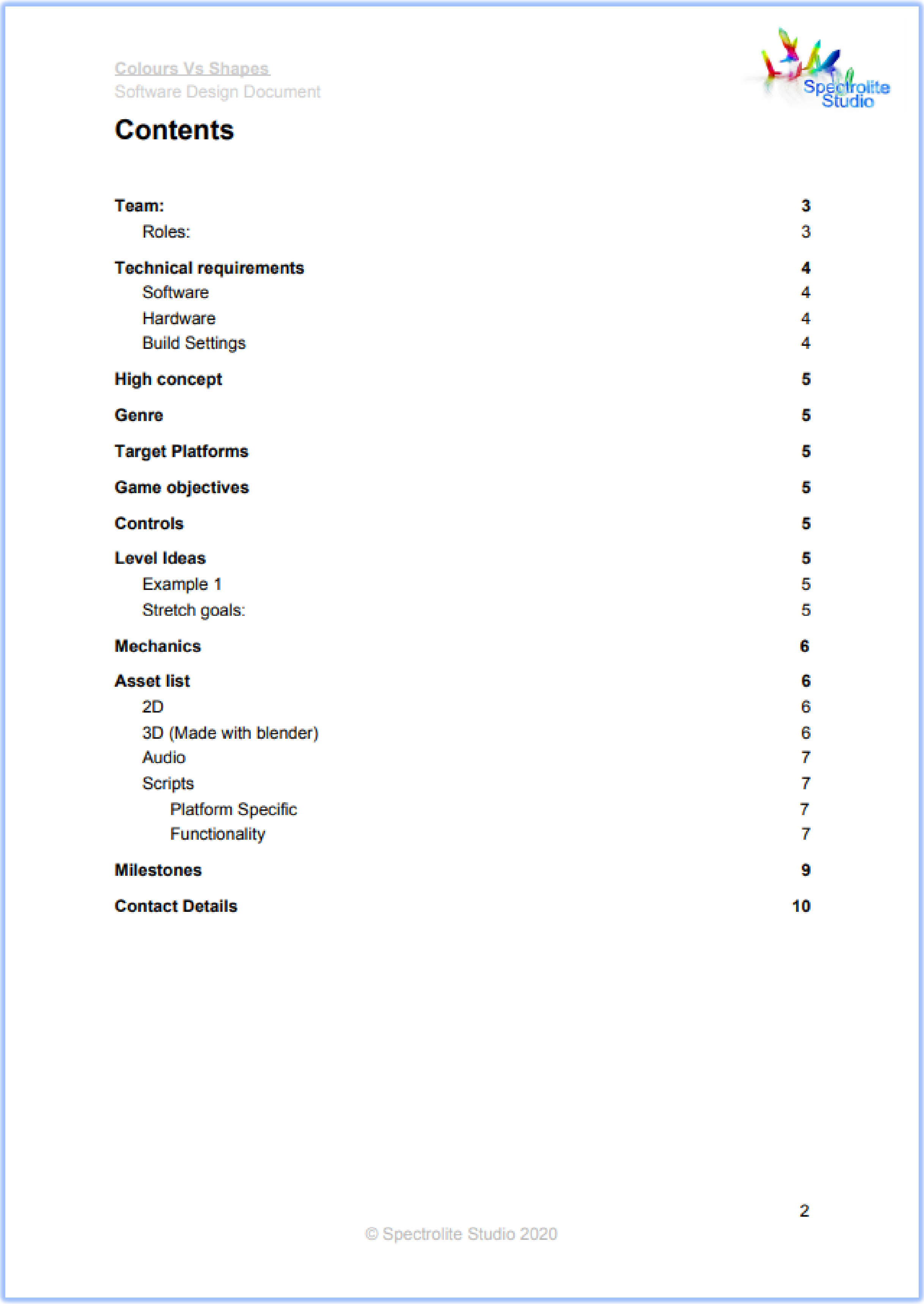
14.0 Table of Appendices

Appendix 1 - Game Design Document........ 27
Appendix 2 - Blog Post 39

Appendix 1 - Game Design Document

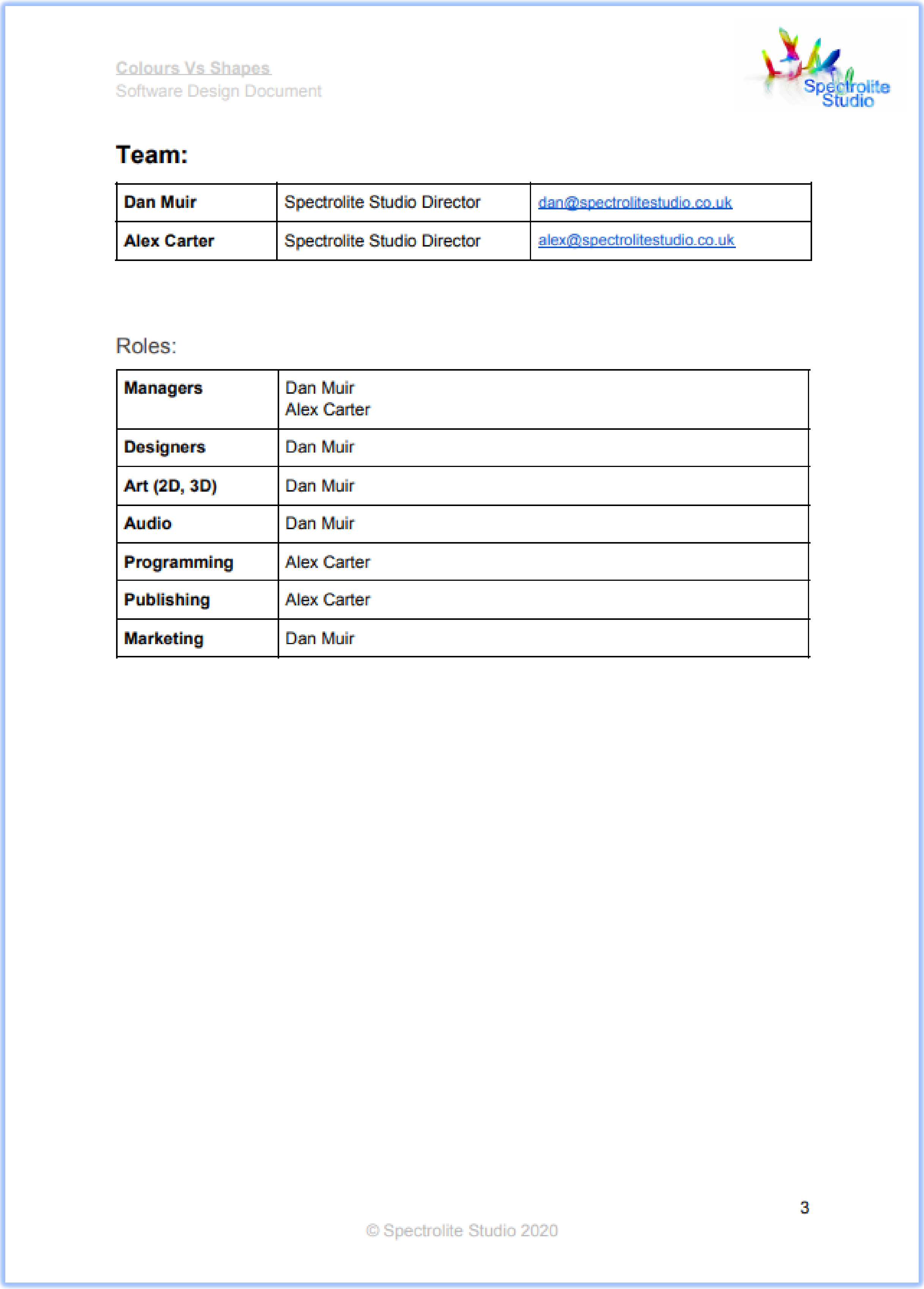
Colours Vs Shapes

A Spectrolite Studio Production
Software Design Document



Contents

Team: **3**
Roles: 3
Technical requirements **4**
Software 4
Hardware 4
Build Settings 4
High concept **5**
Genre **5**
Target Platforms **5**
Game objectives **5**
Controls **5**
Level Ideas **5**
Example 1 5
Stretch goals: 5
Mechanics **6**
Asset list **6**
2D 6
3D (Made with blender) 6
Audio 7
Scripts 7
Platform Specific 7
Functionality 7
Milestones **9**
Contact Details **10**



Colours Vs Shapes
Software Design Document

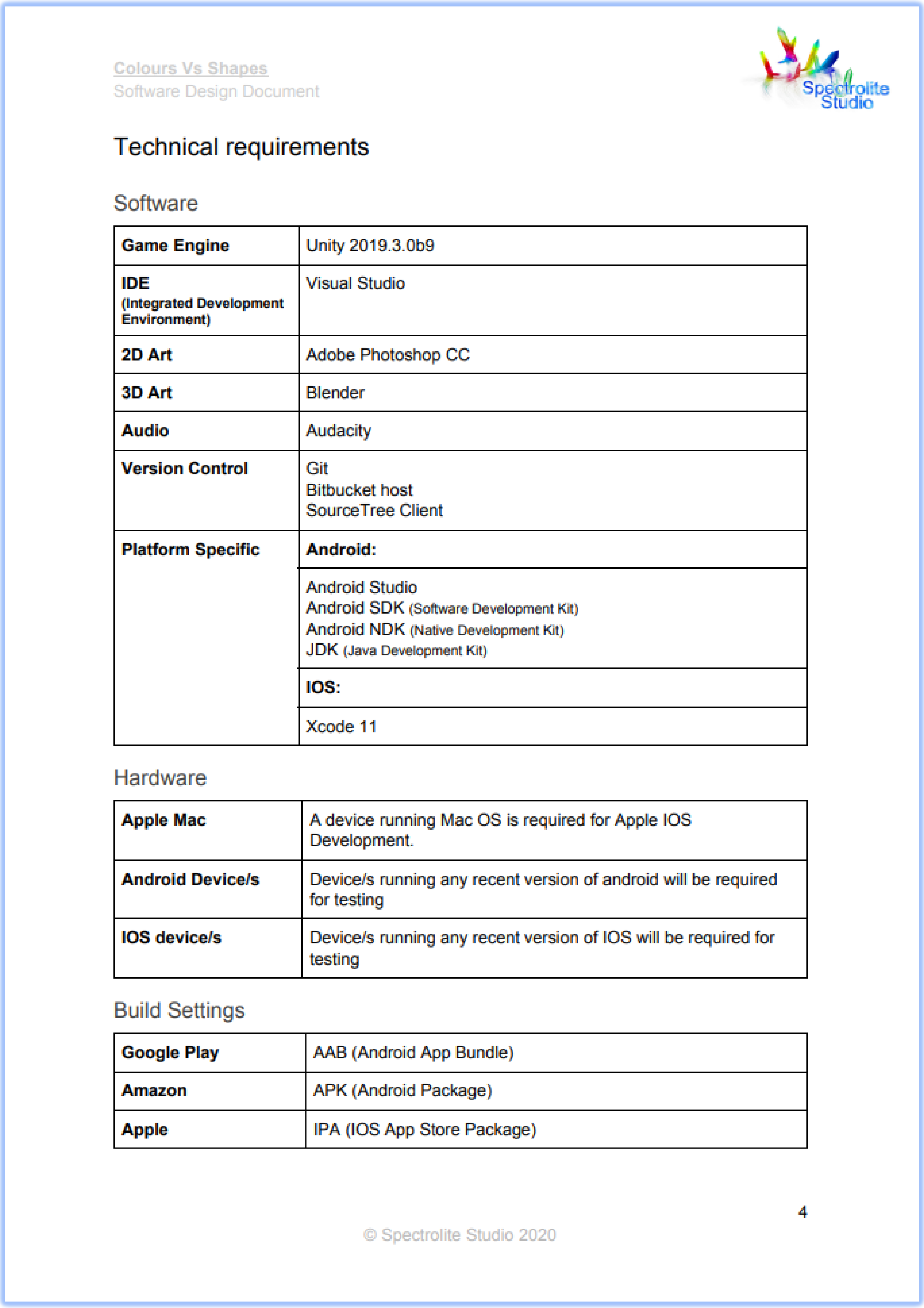
Team:

Dan Muir	Spectrolite Studio Director	dan@spectrolitestudio.co.uk
Alex Carter	Spectrolite Studio Director	alex@spectrolitestudio.co.uk

Roles:

Managers	Dan Muir Alex Carter
Designers	Dan Muir
Art (2D, 3D)	Dan Muir
Audio	Dan Muir
Programming	Alex Carter
Publishing	Alex Carter
Marketing	Dan Muir

3



Colours Vs Shapes
Software Design Document

Spectrolite Studio

Technical requirements

Software

Game Engine	Unity 2019.3.0b9
IDE **(Integrated Development Environment)**	Visual Studio
2D Art	Adobe Photoshop CC
3D Art	Blender
Audio	Audacity
Version Control	Git Bitbucket host SourceTree Client
Platform Specific	**Android:**
	Android Studio Android SDK (Software Development Kit) Android NDK (Native Development Kit) JDK (Java Development Kit)
	IOS:
	Xcode 11

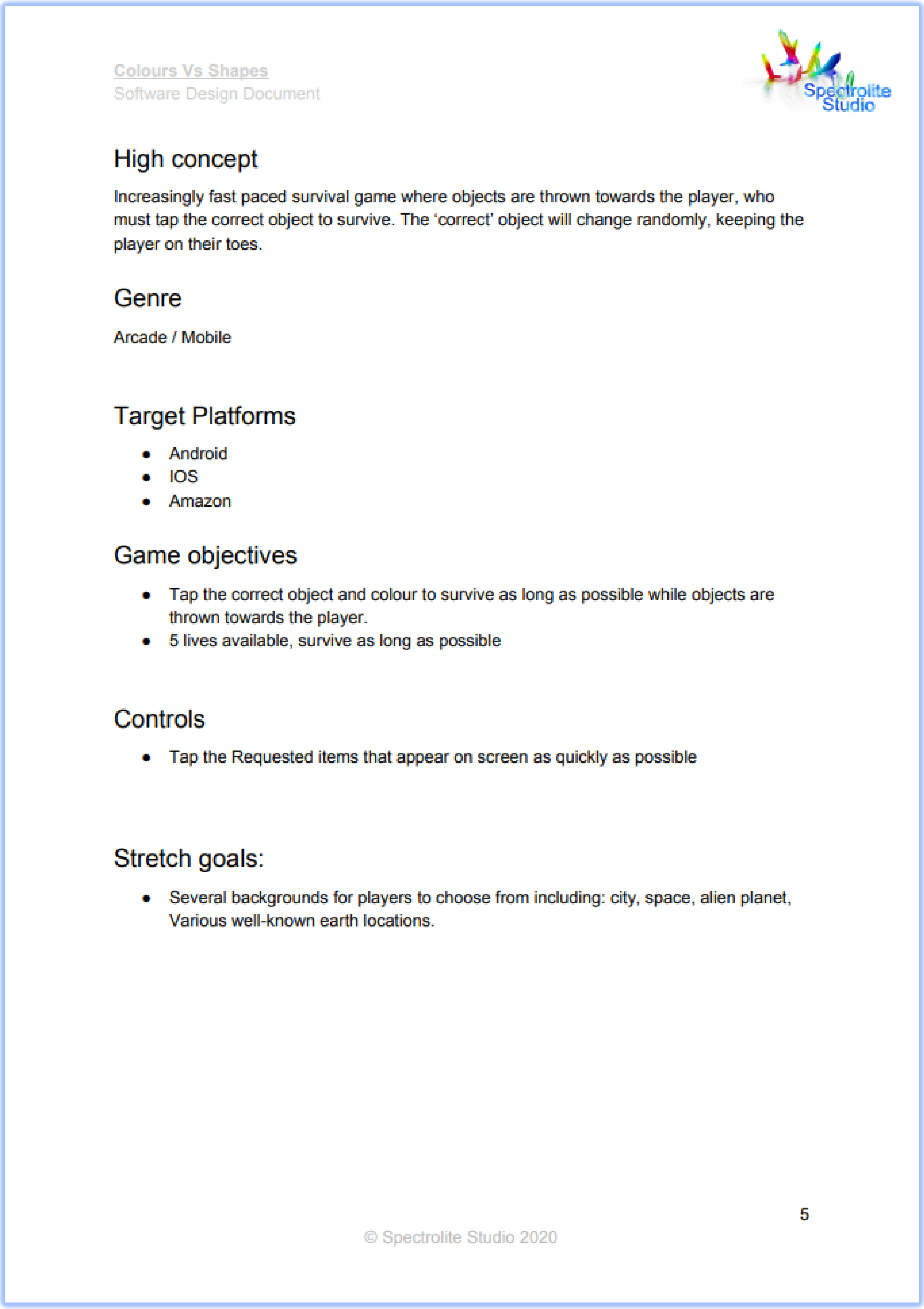
Hardware

Apple Mac	A device running Mac OS is required for Apple IOS Development.
Android Device/s	Device/s running any recent version of android will be required for testing
IOS device/s	Device/s running any recent version of IOS will be required for testing

Build Settings

Google Play	AAB (Android App Bundle)
Amazon	APK (Android Package)
Apple	IPA (IOS App Store Package)

4



Colours Vs Shapes
Software Design Document

Spectrolite Studio

High concept

Increasingly fast paced survival game where objects are thrown towards the player, who must tap the correct object to survive. The 'correct' object will change randomly, keeping the player on their toes.

Genre

Arcade / Mobile

Target Platforms

- Android
- IOS
- Amazon

Game objectives

- Tap the correct object and colour to survive as long as possible while objects are thrown towards the player.
- 5 lives available, survive as long as possible

Controls

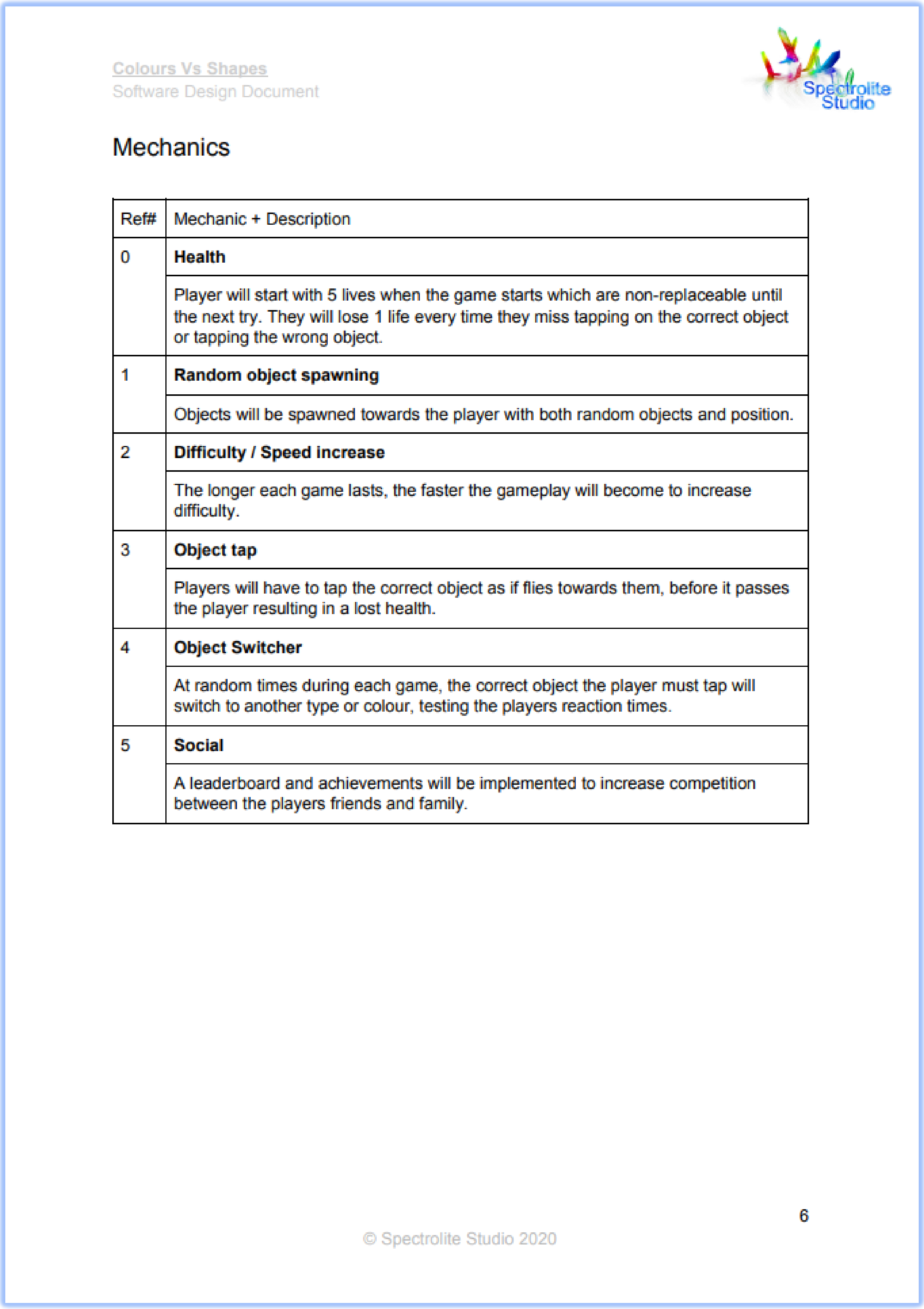
- Tap the Requested items that appear on screen as quickly as possible

Stretch goals:

- Several backgrounds for players to choose from including: city, space, alien planet, Various well-known earth locations.

5

© Spectrolite Studio 2020



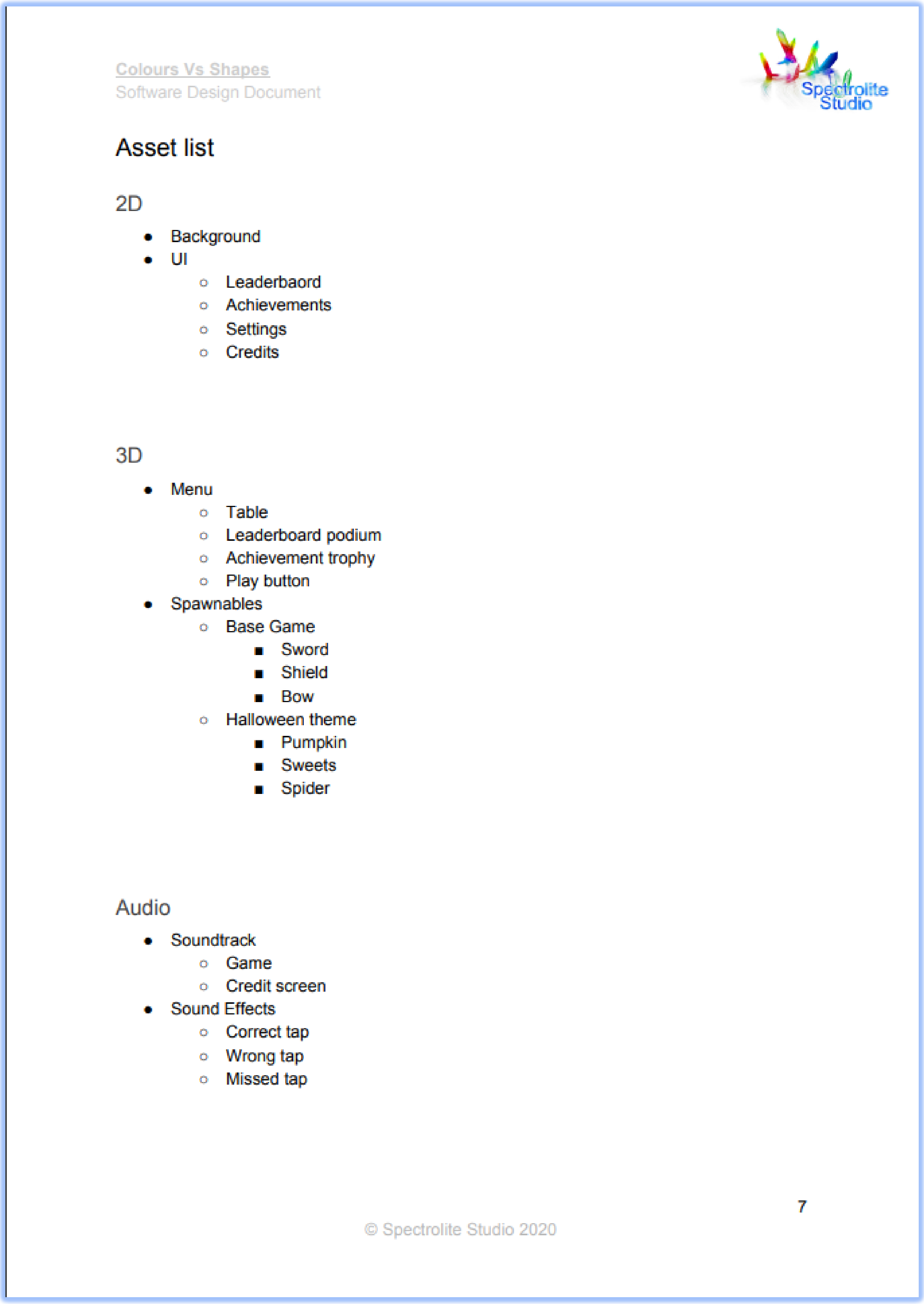
Colours Vs Shapes
Software Design Document

Spectrolite Studio

Mechanics

Ref#	Mechanic + Description
0	**Health**
	Player will start with 5 lives when the game starts which are non-replaceable until the next try. They will lose 1 life every time they miss tapping on the correct object or tapping the wrong object.
1	**Random object spawning**
	Objects will be spawned towards the player with both random objects and position.
2	**Difficulty / Speed increase**
	The longer each game lasts, the faster the gameplay will become to increase difficulty.
3	**Object tap**
	Players will have to tap the correct object as if flies towards them, before it passes the player resulting in a lost health.
4	**Object Switcher**
	At random times during each game, the correct object the player must tap will switch to another type or colour, testing the players reaction times.
5	**Social**
	A leaderboard and achievements will be implemented to increase competition between the players friends and family.

6



Spectrolite Studio

Asset list

2D

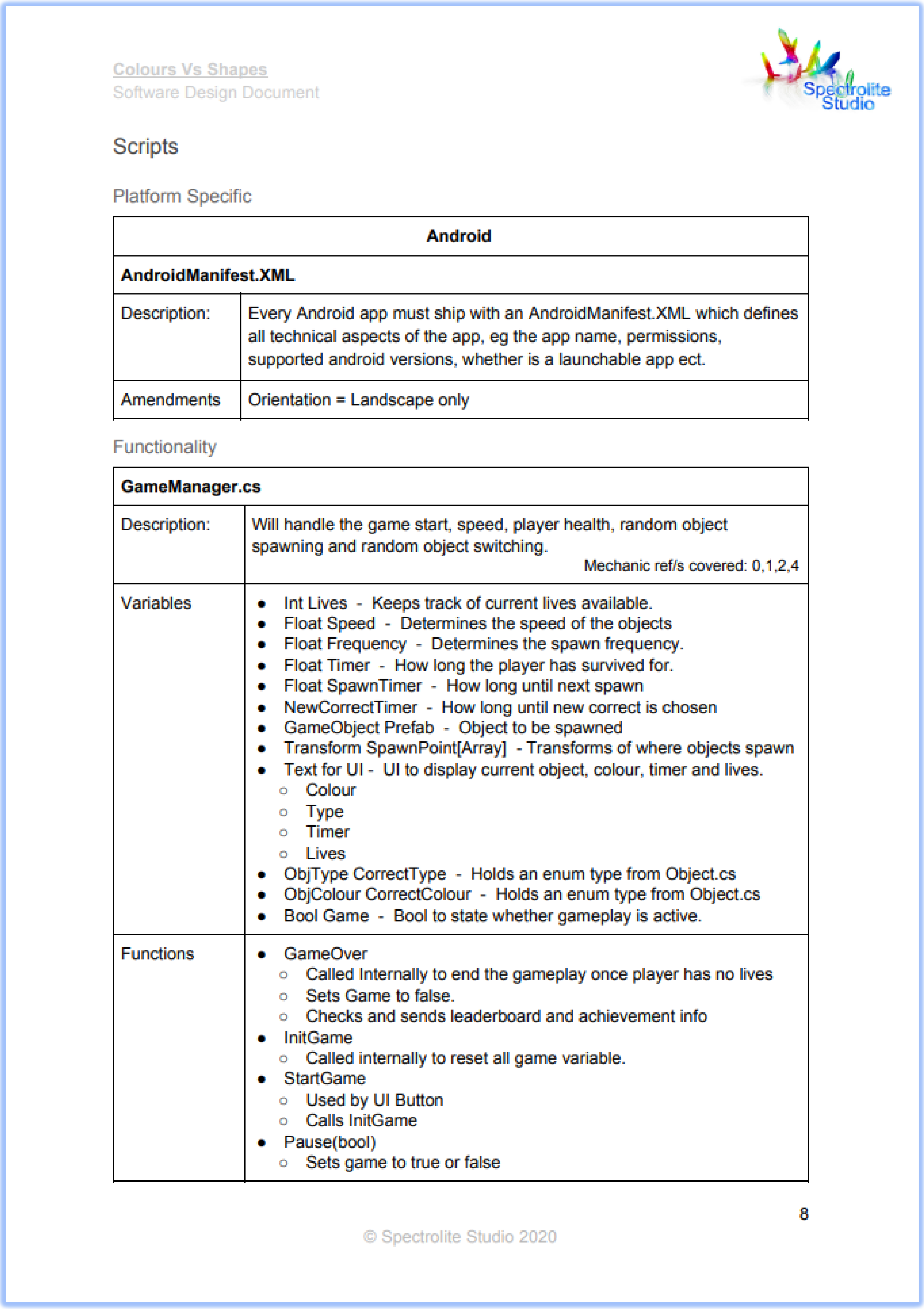
- Background
- UI
    - Leaderbaord
    - Achievements
    - Settings
    - Credits

3D

- Menu
    - Table
    - Leaderboard podium
    - Achievement trophy
    - Play button
- Spawnables
    - Base Game
        - Sword
        - Shield
        - Bow
    - Halloween theme
        - Pumpkin
        - Sweets
        - Spider

Audio

- Soundtrack
    - Game
    - Credit screen
- Sound Effects
    - Correct tap
    - Wrong tap
    - Missed tap



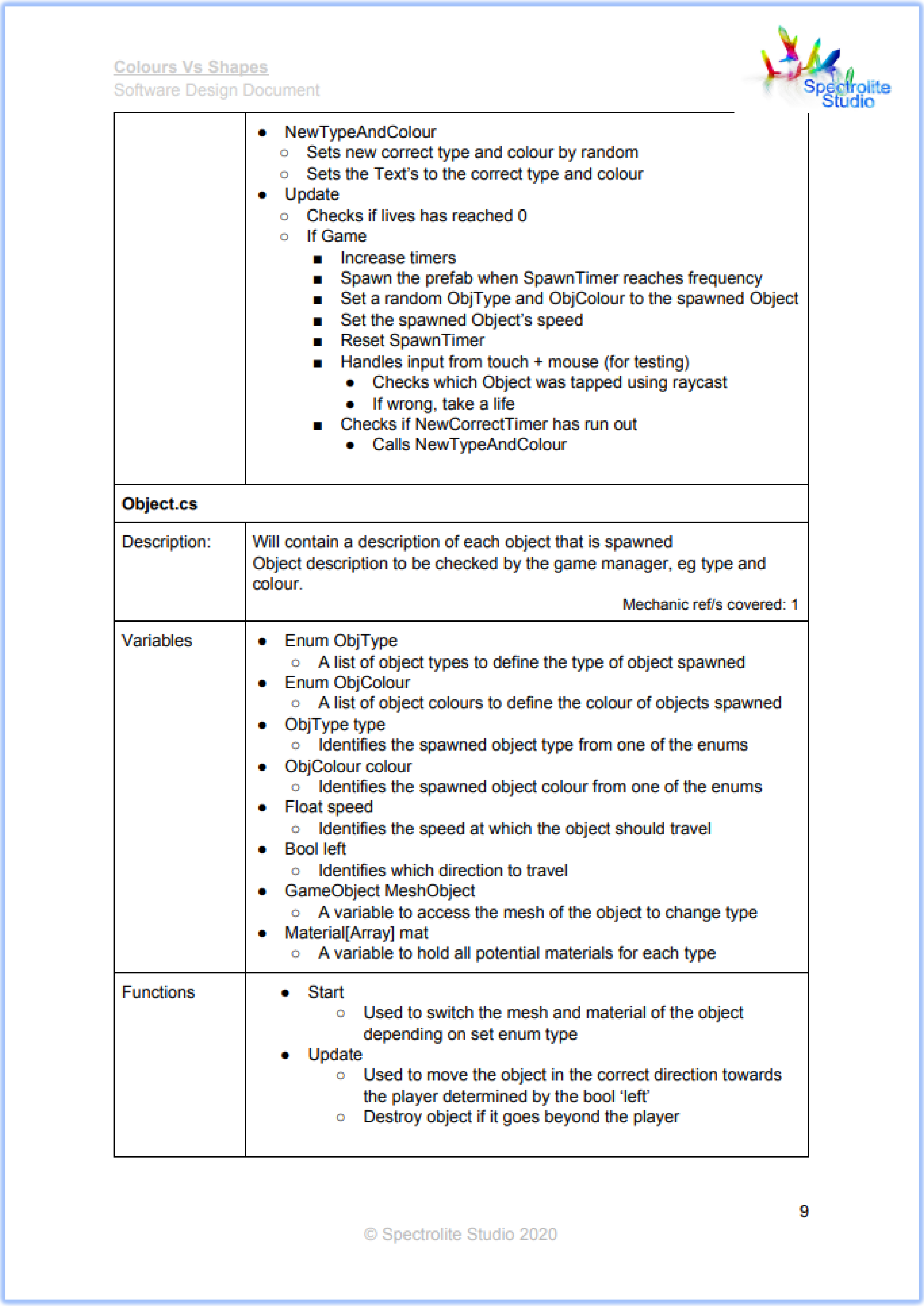
Scripts

Platform Specific

Android	
AndroidManifest.XML	
Description:	Every Android app must ship with an AndroidManifest.XML which defines all technical aspects of the app, eg the app name, permissions, supported android versions, whether is a launchable app ect.
Amendments	Orientation = Landscape only

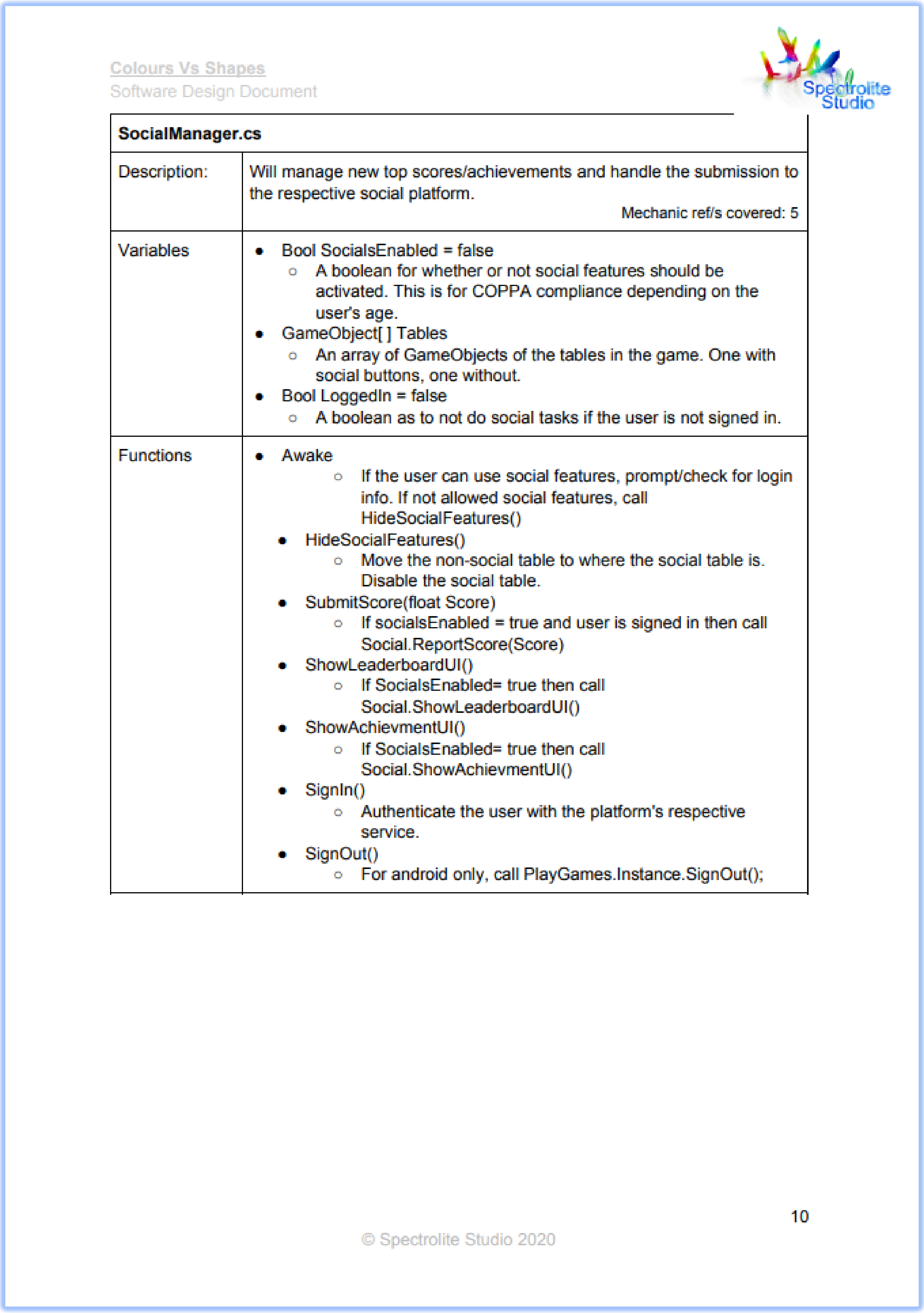
Functionality

GameManager.cs	
Description:	Will handle the game start, speed, player health, random object spawning and random object switching. Mechanic ref/s covered: 0,1,2,4
Variables	• Int Lives - Keeps track of current lives available. • Float Speed - Determines the speed of the objects • Float Frequency - Determines the spawn frequency. • Float Timer - How long the player has survived for. • Float SpawnTimer - How long until next spawn • NewCorrectTimer - How long until new correct is chosen • GameObject Prefab - Object to be spawned • Transform SpawnPoint[Array] - Transforms of where objects spawn • Text for UI - UI to display current object, colour, timer and lives. ○ Colour ○ Type ○ Timer ○ Lives • ObjType CorrectType - Holds an enum type from Object.cs • ObjColour CorrectColour - Holds an enum type from Object.cs • Bool Game - Bool to state whether gameplay is active.
Functions	• GameOver ○ Called Internally to end the gameplay once player has no lives ○ Sets Game to false. ○ Checks and sends leaderboard and achievement info • InitGame ○ Called internally to reset all game variable. • StartGame ○ Used by UI Button ○ Calls InitGame • Pause(bool) ○ Sets game to true or false



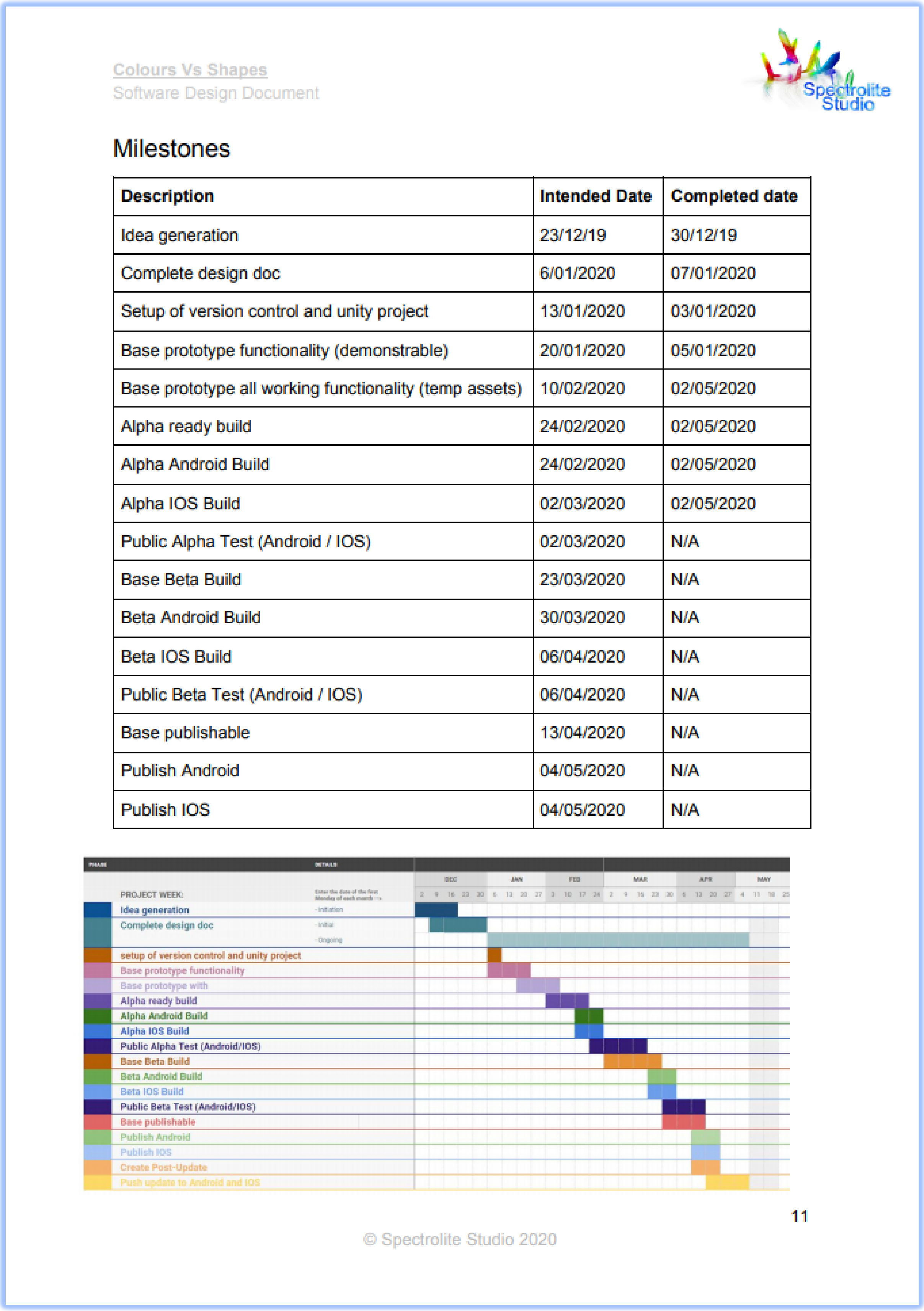
Spectrolite
Studio

	• NewTypeAndColour ○ Sets new correct type and colour by random ○ Sets the Text's to the correct type and colour • Update ○ Checks if lives has reached 0 ○ If Game ■ Increase timers ■ Spawn the prefab when SpawnTimer reaches frequency ■ Set a random ObjType and ObjColour to the spawned Object ■ Set the spawned Object's speed ■ Reset SpawnTimer ■ Handles input from touch + mouse (for testing) • Checks which Object was tapped using raycast • If wrong, take a life ■ Checks if NewCorrectTimer has run out • Calls NewTypeAndColour
Object.cs	
Description:	Will contain a description of each object that is spawned Object description to be checked by the game manager, eg type and colour. Mechanic ref/s covered: 1
Variables	• Enum ObjType ○ A list of object types to define the type of object spawned • Enum ObjColour ○ A list of object colours to define the colour of objects spawned • ObjType type ○ Identifies the spawned object type from one of the enums • ObjColour colour ○ Identifies the spawned object colour from one of the enums • Float speed ○ Identifies the speed at which the object should travel • Bool left ○ Identifies which direction to travel • GameObject MeshObject ○ A variable to access the mesh of the object to change type • Material[Array] mat ○ A variable to hold all potential materials for each type
Functions	• Start ○ Used to switch the mesh and material of the object depending on set enum type • Update ○ Used to move the object in the correct direction towards the player determined by the bool 'left' ○ Destroy object if it goes beyond the player



Spectrolite Studio

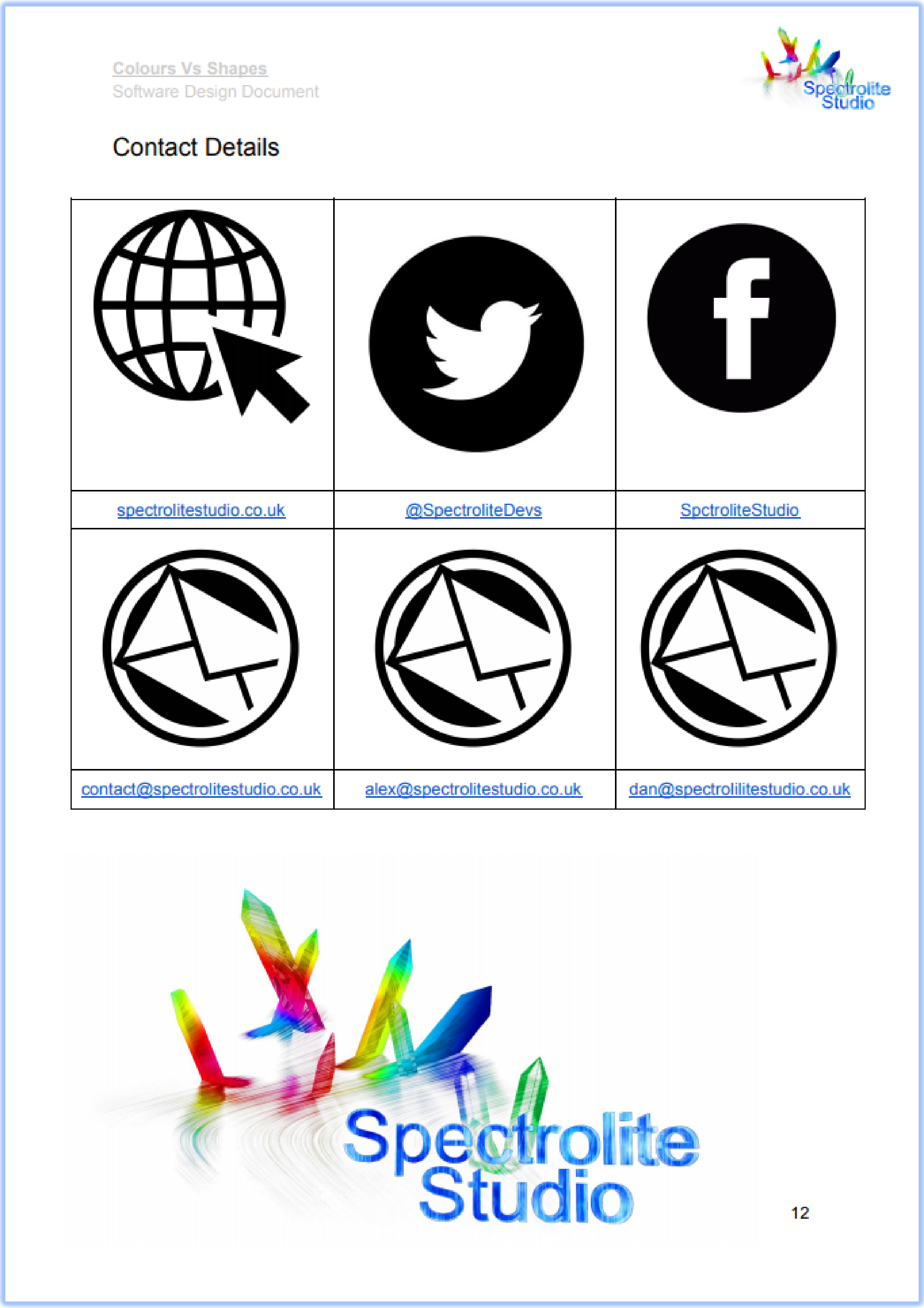
SocialManager.cs	
Description:	Will manage new top scores/achievements and handle the submission to the respective social platform. Mechanic ref/s covered: 5
Variables	• Bool SocialsEnabled = false ○ A boolean for whether or not social features should be activated. This is for COPPA compliance depending on the user's age. • GameObject[ ] Tables ○ An array of GameObjects of the tables in the game. One with social buttons, one without. • Bool LoggedIn = false ○ A boolean as to not do social tasks if the user is not signed in.
Functions	• Awake ○ If the user can use social features, prompt/check for login info. If not allowed social features, call HideSocialFeatures() • HideSocialFeatures() ○ Move the non-social table to where the social table is. Disable the social table. • SubmitScore(float Score) ○ If socialsEnabled = true and user is signed in then call Social.ReportScore(Score) • ShowLeaderboardUI() ○ If SocialsEnabled= true then call Social.ShowLeaderboardUI() • ShowAchievmentUI() ○ If SocialsEnabled= true then call Social.ShowAchievmentUI() • SignIn() ○ Authenticate the user with the platform's respective service. • SignOut() ○ For android only, call PlayGames.Instance.SignOut();



Spectrolite Studio

Milestones

Description	Intended Date	Completed date
Idea generation	23/12/19	30/12/19
Complete design doc	6/01/2020	07/01/2020
Setup of version control and unity project	13/01/2020	03/01/2020
Base prototype functionality (demonstrable)	20/01/2020	05/01/2020
Base prototype all working functionality (temp assets)	10/02/2020	02/05/2020
Alpha ready build	24/02/2020	02/05/2020
Alpha Android Build	24/02/2020	02/05/2020
Alpha IOS Build	02/03/2020	02/05/2020
Public Alpha Test (Android / IOS)	02/03/2020	N/A
Base Beta Build	23/03/2020	N/A
Beta Android Build	30/03/2020	N/A
Beta IOS Build	06/04/2020	N/A
Public Beta Test (Android / IOS)	06/04/2020	N/A
Base publishable	13/04/2020	N/A
Publish Android	04/05/2020	N/A
Publish IOS	04/05/2020	N/A



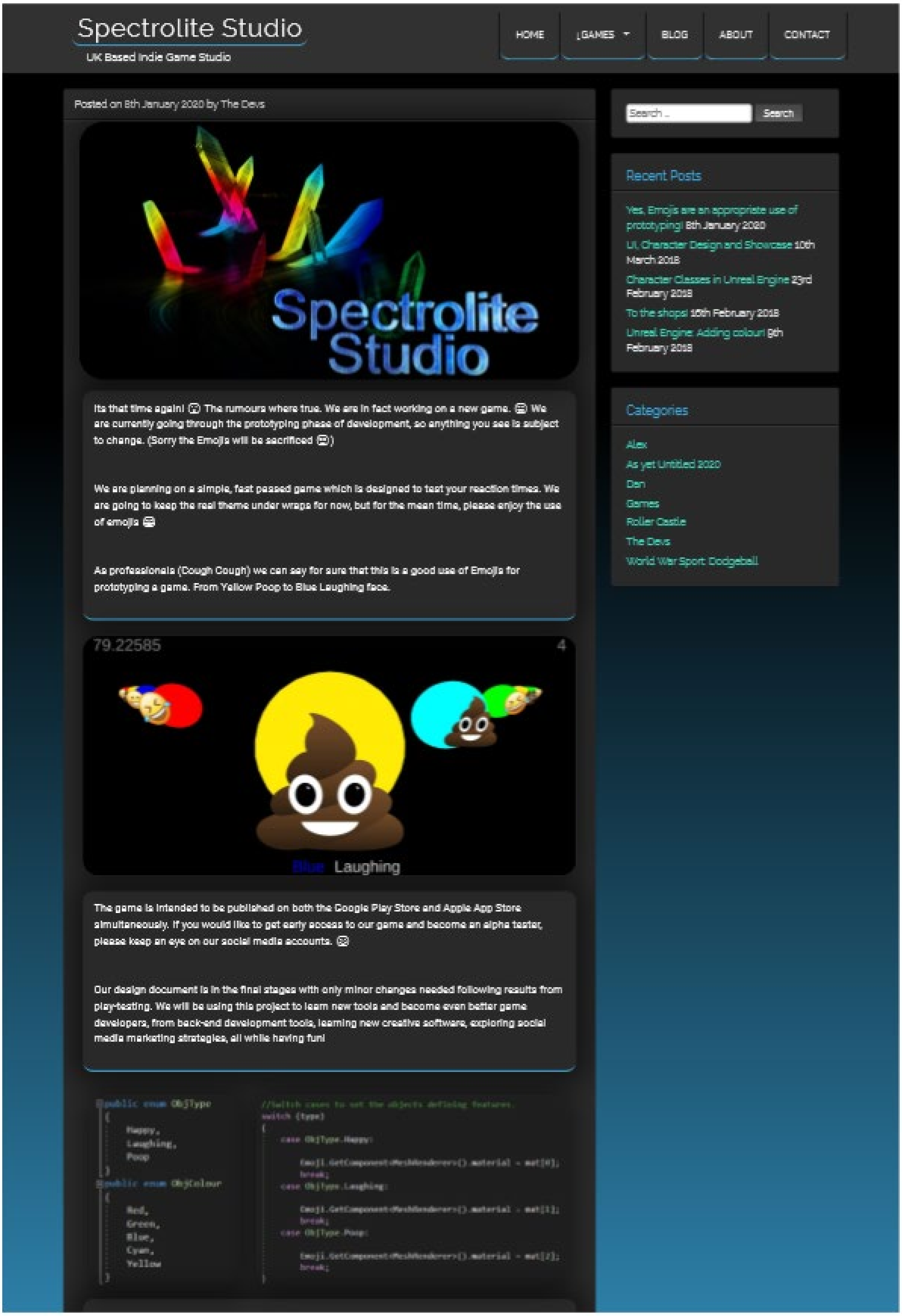
Colours Vs Shapes
Software Design Document

Contact Details

🌐	🐦	f
spectrolitestudio.co.uk	@SpectroliteDevs	SpctroliteStudio
✉	✉	✉
contact@spectrolitestudio.co.uk	alex@spectrolitestudio.co.uk	dan@spectrolilitestudio.co.uk

12

Appendix 2 - Blog Post



Posted on 8th January 2020 by The Devs

Its that time again! 😢 The rumours where true. We are in fact working on a new game. 😂 We are currently going through the prototyping phase of development, so anything you see is subject to change. (Sorry the Emojis will be sacrificed 😢)

We are planning on a simple, fast passed game which is designed to test your reaction times. We are going to keep the real theme under wraps for now, but for the mean time, please enjoy the use of emojis 😂

As professionals (Cough Cough) we can say for sure that this is a good use of Emojis for prototyping a game. From Yellow Poop to Blue Laughing face.

The game is intended to be published on both the Google Play Store and Apple App Store simultaneously. If you would like to get early access to our game and become an alpha tester, please keep an eye on our social media accounts. 😂

Our design document is in the final stages with only minor changes needed following results from play-testing. We will be using this project to learn new tools and become even better game developers, from back-end development tools, learning new creative software, exploring social media marketing strategies, all while having fun!

```
public enum ObjType
{
    Happy,
    Laughing,
    Poop
}
public enum ObjColour
{
    Red,
    Green,
    Blue,
    Cyan,
    Yellow
}
```

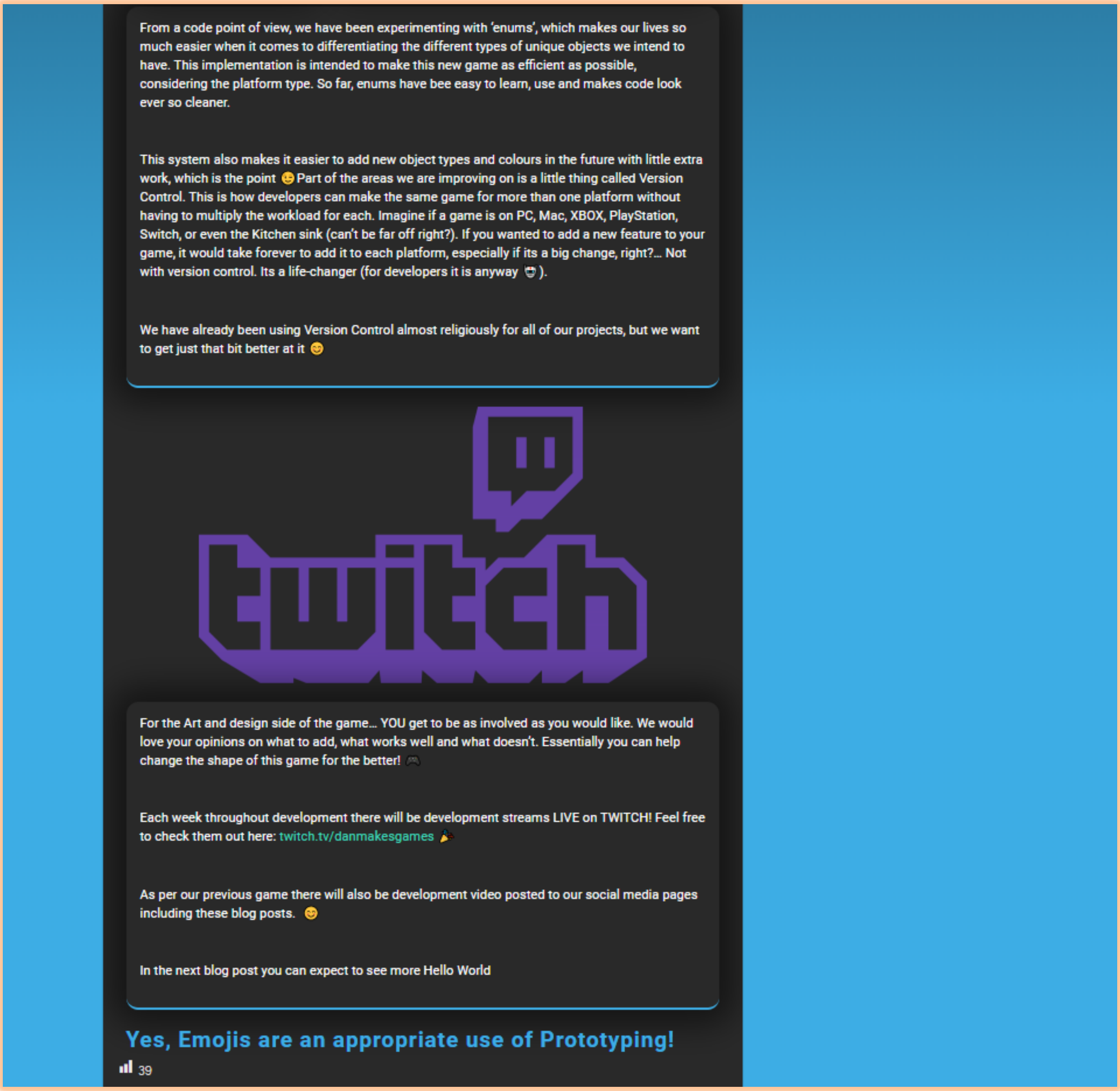
```
//Switch cases to set the objects defining features.
switch (type)
{
    case ObjType.Happy:

        Emoji.GetComponent<MeshRenderer>().material = mat[0];
        break;
    case ObjType.Laughing:

        Emoji.GetComponent<MeshRenderer>().material = mat[1];
        break;
    case ObjType.Poop:

        Emoji.GetComponent<MeshRenderer>().material = mat[2];
        break;
}
```

Recent Posts

Yes, Emojis are an appropriate use of prototyping! 8th January 2020
UI, Character Design and Showcase 10th March 2018
Character Classes in Unreal Engine 23rd February 2018
To the shops! 16th February 2018
Unreal Engine: Adding colour! 9th February 2018

Categories

Alex
As yet Untitled 2020
Dan
Games
Roller Castle
The Devs
World War Sport: Dodgeball



From a code point of view, we have been experimenting with 'enums', which makes our lives so much easier when it comes to differentiating the different types of unique objects we intend to have. This implementation is intended to make this new game as efficient as possible, considering the platform type. So far, enums have bee easy to learn, use and makes code look ever so cleaner.

This system also makes it easier to add new object types and colours in the future with little extra work, which is the point 😉 Part of the areas we are improving on is a little thing called Version Control. This is how developers can make the same game for more than one platform without having to multiply the workload for each. Imagine if a game is on PC, Mac, XBOX, PlayStation, Switch, or even the Kitchen sink (can't be far off right?). If you wanted to add a new feature to your game, it would take forever to add it to each platform, especially if its a big change, right?... Not with version control. Its a life-changer (for developers it is anyway 🤓).

We have already been using Version Control almost religiously for all of our projects, but we want to get just that bit better at it 😊

For the Art and design side of the game... YOU get to be as involved as you would like. We would love your opinions on what to add, what works well and what doesn't. Essentially you can help change the shape of this game for the better! 🎮

Each week throughout development there will be development streams LIVE on TWITCH! Feel free to check them out here: twitch.tv/danmakesgames 🎉

As per our previous game there will also be development video posted to our social media pages including these blog posts. 😊

In the next blog post you can expect to see more Hello World

Yes, Emojis are an appropriate use of Prototyping!

📶 39