Is it possible to create an algorithm that can accurately predict a set of shots a director should employ before creating a film?

Jonathan Byrne
BYR19702034

19702034@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the

Degree of MComp Games Computing

Supervisor: Phil Carlisle

May 2023

Acknowledgements

I wish to thank my supervisor Phil Carlisle for his guidance during this project, together with my parents and brother for helping with the structure of this dissertation and their advice.

Abstract

This dissertation describes the development of a recommendation system for film shots with the aim of being used by both laypeople and professional directors. This document contains an introduction as to why this system was created, a review of literature similar to this project and how it influenced the aims and objectives, the requirements for this project based on the literature review and the aims and objectives, an outline of the design, tools and the software methodologies used, a history of the implementation of this project followed by the results of testing and finally the conclusion gathered from this project.

The Github repository for this project can be downloaded following this link: https://github.com/JonBYR/Year3Project.git

Keywords: Unity, recommendation system, SCRUM, Agile methodologies, Scrumban, waterfall, cinematography, previs, storyboard, Kanban, User Experience Design, User Interface

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Rationale

Parts of this introduction have been taken from a prior assignment (Byrne, 2022).

In the world of film production, there exists a developmental pipeline for film creation. Generally, this can be thought of as existing in three specific stages: pre-production, production and post-production. According to Case “Digital imaging technology has emerged in the film industry in three separate fields: film production (image capture), post production and cinema distribution” (Case, 2013, 18). However, there is a lack of commercial software that currently exists in the pre-production phase, where techniques such as previsualisation occur. Previsualisation (or previs) can be defined in many ways. One example could be a shot sheet that “provides a list of shot descriptions for each camera” (Shyles, 2007, 454) or it could be a storyboarding process that a director could use to plan aspects of the film such as plot, setting and, in the case of this project, cinematography. Shyles states that a storyboard “shows the main shots of a story” (Shyles, 2007, 425).

This project aims to develop a software to aid directors in picking a set of cinematic shots in the pre-production phase by evaluating the most common shots in pre-existing films, with these common shots then being recommended to the user. The average cost of producing a film “has reached $50.4 million in 1995” (Ravid, 1999, 464) and the cost of production can only be expected to have risen from this point. Even indie films such as *Everything Everywhere All At Once* (Kwan, Scheinert, 2022) can have high production costs, with the film requiring

$25 million to make (The Numbers, 2022). Reasons for high costs include “talent and crew salaries; rental of equipment and studio space; fees for location shoots” (Shyles, 2007, 438) to name but a few. Part of this cost can come as a result of the need to reshoot various scenes, with one factor causing reshooting being the incorrect type of shot being selected initially. The more a director needs to reshoot, the more costs for renting equipment etc increase. The aim of this project is that by outlining a sequence of shots in an ordered visual storyboard, the number of reshoots a director may need to undertake can be reduced as a better visual plan of how a film will look in production, rather than in post-production will be obtained. As more time is spent planning shots in previs this should lead to reduced spending in the production stage as the need for reshooting scenes should be reduced.

Another factor informing the creation of this project is that the software used for previsualisation will also be a help to guide laypeople previously unfamiliar with film practice to gain a visual representation of different shot types that they can then use in the planning of their own films and cinematics. Films consist of different kinds of shots, such as “framing shots and function shots” (Brown, 2016, 60). Framing dictates how much of a subject filmed is included in the shot while function develops the context of the scenes. Framing shots can be defined using terms such as full shot, where a person’s whole body is in frame, while function can be defined using terms such as an establishing shot, which establishes the location of a scene. An overview of framing shots is shown in Figure 1 while a type of function shot, the establishing, is shown in Figure 2.

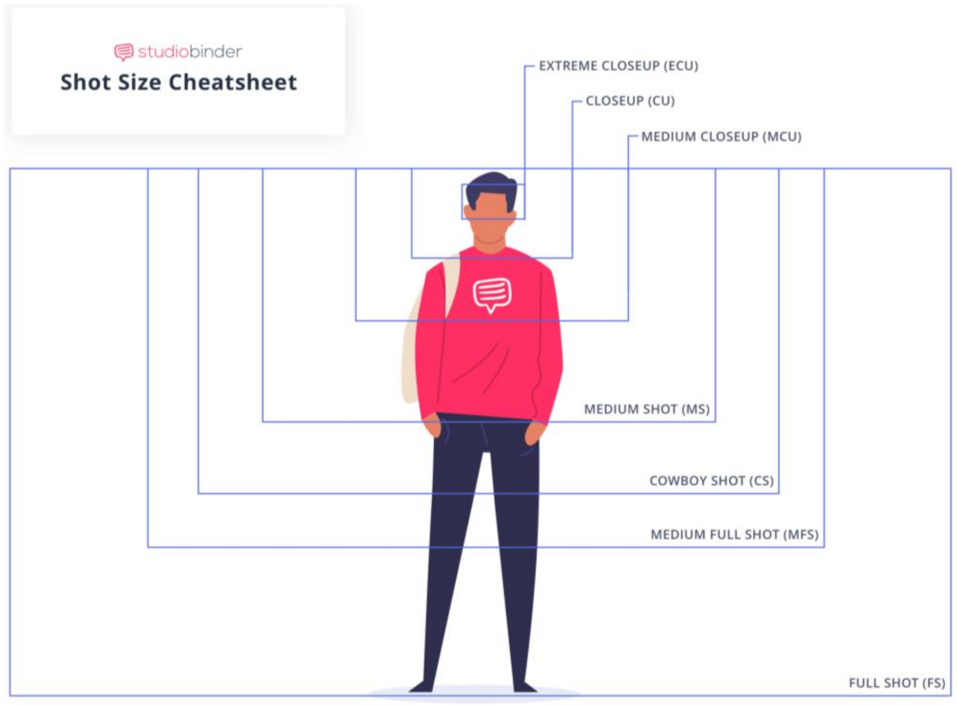


Figure 1: A figure to show different framing shots (Studiobinder, undated)

Note that a cowboy shot and medium full shot are considered the same (Studiobinder, 2020).

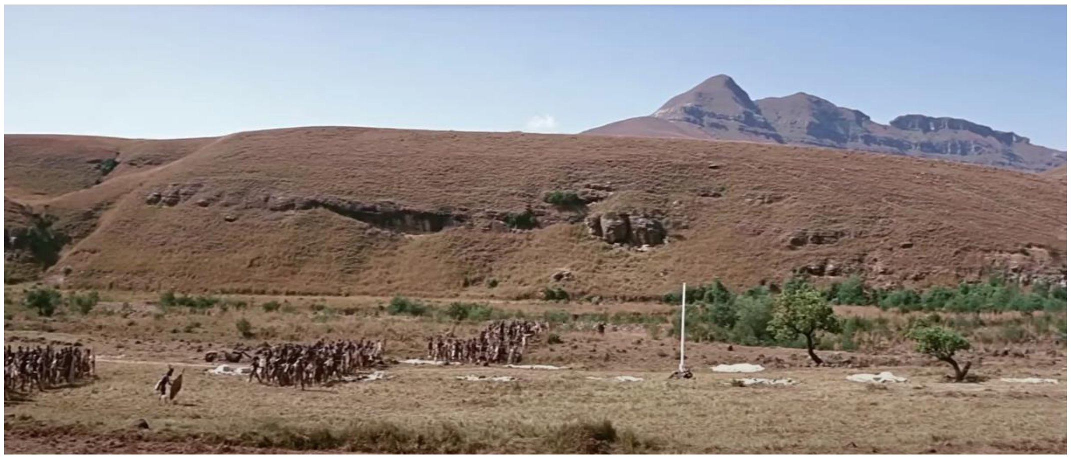


Figure 2: A figure showing an establishing shot used in the film Zulu (Endfield, 1964)

Whilst types of shots may be well understood by existing filmmakers, those that are not as familiar with this terminology may not be able to accurately plan their storyboarding during previsualisation, due to being less familiar with the wide array of cinematic shots that can be employed. Having a software with a visual guide to use during the pre-production stage will help laypeople to properly structure a film scene with the appropriate shots needed for production which will therefore lead to them producing better quality films.

1.2 Dissertation Structure

This dissertation will contain six further chapters. Chapter 2 is a review of literature and how this influenced both the project and the aims and objectives. Chapter 3 details the requirements this project needed to fulfil. Chapter 4 explains how the project was planned, including the software methodologies used, the design of the project, the toolsets used and how the project was tested. Chapter 5 details how the project was implemented over time. Chapter 6 details the results of testing via black box and user tests. Chapter 7 summarises and reflects on the project as a whole.

Chapter 2

Literature Review and Aims and Objectives

2.1 Literature Review

This chapter builds from the prior literature review at the interim stage (Byrne, 2023).

As mentioned by Brown earlier, films consist of a series of different shots, which can be categorised as either framing or function shots. Whilst the different types of shots that exist may be understood, there needs to be a way to quantify what shots are present in a film and the sequence in which they appear. Ronford et al. (2015) developed a language called PROSE, designed as “a formal language for describing shots visually” (1) which was used to divide a scene into its constituent shots and to then link them to their respective timeframes (6). The specific aim of PROSE was to make cinematic shots easily understandable for the user, breaking down each scene into its relative shots as described by Brown. However, there are some issues identified with the use of this language. Whilst it gives a detailed summary of specific shots in a film, the paper itself only linked four examples of films that were broken down using PROSE. Also, PROSE can only be used to break down the shots of existing films and not to create shot sequences for new films. Wu et al. (2018) developed this concept further by creating a new language called FEP (Film Editing Patterns). FEP analysed styles from annotated films and developed this further by incorporating this language into a tool for “shooting and editing 3D animated sequences” (2), done by “selecting a framing (i.e. a full shot) from a framing database…on the basis of the 3D position of actors at the cut point” (13). Again, this study used an existing edited sequence rather than a recommendation for a completely original film. However, FEP does demonstrate the same functionality proposed by this project: a recommendation system of shots for film creation. The way Wu et al. tested their system was via the use of a 3D software. Their aim was to return only the shot list, with no visual indication of how the shots would look. It may be more advisable to demonstrate this sequence in a software to give users a more informed choice as to whether the output is a reasonable vision for this specific genre.

While there may not be a specific commercial software for previsualisation, there have been attempts to create or use existing software to model a film in its pre-production stage. Nitsche (2008) had discussed the use of game engines such as Unity or Unreal as potential vehicles for previsualisation in film. What was discovered was that the use of real time rendering "allows for changes in the action and visualization during runtime and at any stage in the process" (10), meaning that during the pre-production phase any core storyboarding changes could be made in real-time, thus it would be easier to see how these changes would affect the overall product. This was partially done in their project ShotBox, which would allow the director "with two button presses activate a specific camera such as a close up of the face, or an over-the-shoulder shot" (9). Whilst this system relies on the user knowing the existing shot type they wish to implement, this can be utilised as a way to test what a shot sequence will look like by using the real-time capabilities of a game engine linked with the shot sequence the user has been given. Just as ShotBox uses the camera of the specific engine to demonstrate what the shots will look like, this project will similarly aim to use the camera as a device to replicate the shots that are provided to the user.

This methodology is further supported by Goussencourt et al. (2015) who argue that the game engine Unity has a "built in interface that allows an operator to edit in real time” (3) but also noted that the system can be given custom control “relative to your scripts into the editor interface” (7). While this study uses information fed into Unity via camera recorded footage rather than film stills, it would seem that the software could be modified to give the user more control pending on the degree of functionality needed. In this project enough control in the system needs to be given to the user to enable them to easily move the camera in order to replicate the shots displayed to them, rather than having this information fed directly into the camera as it was in Nitsche’s study.

Wu et al. had also earlier implemented a different language called Patterns (2015). The aim of this language was to “represent the semantics of framing and frame sequencing over a number of shots” (3) which again links back to what Brown describes. Using Patterns the system generates a shot sequence from a user’s specification, for instance utilising other types of shots such as a point of view shot. From there Patterns would search through a database and return anything that matched the specific framing properties outlined. This is similar to what is proposed in this project as there will also be a requirement for the system to return a sequence of shots. However, the system will ask the user to specify other aspects of film such as genre, rather than relate solely to specific framings as in Wu’s study.

Shots are applied to a film to give emphasis to certain events. A close up, as Heiderich describes, means “character’s actions are more intimate and impacting” (2012, 8). Insert shots can be used to feed information to a viewer i.e. that a particular establishment is “run down and creepy” (Brown, 2016, 67). As a result, shots are chosen in terms of narrative importance. Wu mentions that the FEP language was used in conjunction with a tool for the creation of 3D film sequences. Specifically, it was used with human participants who were told the overall outcome “should be coherent and aesthetically pleasing, in their own judgement” (18). The artefact should therefore also be able to produce a shot sequence that adheres to these principles, i.e. that a shot should be used for narrative effect. This could be achieved by using the same narrative context for each scene for both analysis and replication purposes, meaning that the shot sequences used as input data and the shot sequence subsequently generated from this will be consistent with one another. This would, however, limit the scope of the project itself, as the project would only be able to create a certain subset of shot sequences rather than a set of shot sequences that match with different scenarios in a film, such as a break-up or shoot out scene.

One thing to note from Wu’s testing with FEP is how “amateur video editors were able to make use of FEP to create sequences that had similar shots to professionals” (20). The benefit of Wu’s study, therefore, is that a recommendation system is created. Robillard et al. describes these as systems to “help people … make decisions where they lack experience” (2009, 1) and, while this paper relates to software engineering, the principle remains the same, helping amateur film makers who lack experience in film techniques. Recommendation systems are normally achieved using rating systems that put “items most valuable to the user” to the forefront (6). The aim of this artefact is to produce a recommendation system that delivers a list of the most valuable and beneficial items to the user, these being the shots that are most used. The aim of this list will be to inform users how to best structure their films based on pre-existing examples.

In order to generate the shot list, there needs to be input data of some kind that will be handled by the algorithm. One avenue that could be explored would be a machine learning implementation. Vacchetti et al. proposed a way to identify shot types, such as the medium shot, by training three separate VCGs “a convolutional neural network developed by Visual Geometry Group based in Oxford University" (2022, 7) on a dataset of film shots and afterwards combine the predictions with an MLP (multilayer perceptron) classifier (2022, 7). However, they admit that the results found after testing had accuracies of "78%" (2022, 16) which, while good "should be higher in the eyes of the user" (2022, 16). If the accuracy has around a 20% degree of error, then this is an issue. There is a potential that the user will be fed incorrect information by the model predicting the wrong type of shot, which may not be obvious to a layperson. Therefore, it may be that in order to ensure maximum accuracy in this system, a trained human with good knowledge of film shots is used instead to create the input data for the system, rather than a machine learning model to classify shots as the input data.

2.2 Aims & Objectives

The finalised aim for this project was established in the interim stage "Is it possible to create an algorithm that can accurately predict a set of shots a director should employ before creating a film?" (Byrne, 2023) and the finalised objectives are outlined with SMART justification in Table 1:

Table 1: Justification on why each objective is SMART

Objective	**Specific**	**Measurable**	**Achievable**	**Relevant**	**Timebound**
Learn the types of shots that exist in films	At the end of this objective the human researcher should identify film shot types	Learning the types of shots will come from reviewing film related material similar to Brown (2016)	Materials to learn shots exist in sources such as Brown (2016) or online videos such as Studiobinder (2020)	In order to create an algorithm for film shots the types of shots must be specified to enable the algorithm to process them	This objective must be achieved by December 22nd

Analyse the shot types of 10 films to compile as data for the algorithm	At the end of this objective the algorithm should have film data from analysed films	Once 10 films are analysed this objective is evidently achieved	Analysing films should be easily done from knowledge gleamed from the prior objective	In order for the algorithm to predict a shot list it needs a form of input data on which to base its judgement	This objective must be achieved by December 29th
Implement a system to take user input and output	This algorithm needs an input and output system incorporate in some way	If users can give input and receive a relevant output, this objective is achieved	Since both the type of input and output is known as well as the language, all that is needed is to research how this can be created looking at existing documentation.	Since this software for film is created for novices and professionals alike and film is an expressive medium, users should be allowed "to select the topic, content or issue" (Hobbs, 2019, 7) to facilitate creative freedom. In this case the user input is genre and the output is a shot list relating to that genre.	This objective must be achieved by January 13th

Display relevant images to illustrate what shots will look like to user	The output is required to have a visual component attached to it	For shots that are outputted, a relevant image must be attached to it, similar to Figure 2	Research will also be needed on how to display images to users, which can be gathered from online resources	Images should also be included to further a user's understanding of how shots will look	This objective must be achieved by February 20th
Decide on a relevant UI for user interaction	There are different UI's that could be extended as a base for this project, such as Unity (Unity Technologies, 2023) or Unreal (Epic Games, 2022) so a definitive choice is required	Once a UI is decided upon and extended this objective is fulfilled	Information on both engines can be researched online	As users are testing this project, UXD principles must be adhered to and therefore a relevant UI is needed	This objective must be achieved by March 7th
Test artefact on users to measure how effective it is in achieving the aim	This objective means that user testing is required to test the aim	This objective is fulfilled when enough users are found and their data analysed to see if the results	Test participants can be sourced internally at the University of Lincoln	Since the research aim poses that a hypothetical director would use this algorithm it must therefore be tested on	This objective should be completed by April 16th

		satisfy the aim		human participants	

Chapter 3

Requirements Analysis

Based on the aims and objectives, together with the research gathered for this project, a list of requirements was created. One common link in many of the papers researched was that they were creating a software that utilised aspects of a UI and thus needed to facilitate a good user experience. To achieve this the artefact created would also need to provide a good user interface which would follow the principles set out by Nielson (1994). Nielson explains that systems need “words, phrases, and concepts familiar to the user, rather than internal jargon”. This software is designed with the aim that any layperson can understand its functionality. Therefore, when outputting the shot list the first requirement must be:

- Create an output that is easily understandable to any user

As mentioned in the literature review, certain artefacts such as FEPs were created as recommendation systems for testers to use with no requirement for a user to adhere to what the system had generated. It could be used instead to give advice to users who were less experienced or to users open to new ideas. By creating software that allows for more creative freedom Resnick et al. explains “Enhanced interfaces could enable more effective searching of intellectual resources, improved collaboration among teams, and more rapid discovery processes” (1) with the latter two being important for an industry with multiple workers and potentially having tighter schedules. Therefore, another requirement for this software should be:

- Allow users to experiment with the generated output

Finally, the aims and objectives mention that the user needs a relevant UI both to work with as well as to use to input data, in this case genres, into the system to generate an output. In prior studies, the game engine Unity was used, utilising its camera system to frame shots. Since this project is also working with cinematography, it may be an idea to use the existing Unity interface which has already proven successful (it was reported that "2.8B monthly active end-users who are engaging with content created or operated by Unity solutions in 2020." (Unity, 2021) and has ways to extend its UI functionality via its UnityEditor API (Unity, undated). The final requirement therefore would be:

- Use Unity's UI to allow user input

On the non-functional end of this project, there needs to be a way to store information about pre existing films and the shot types that they use. To keep the input data consistent, a control variable needs to be introduced, that being in this project, shot lists having the context of a shoot out sequence. This is the data that is needed to help the algorithm make an informed choice about the shot list it needs to output for the user when a specific genre is input (as an independent variable). On the surface, this sounded like a database problem. A database system is defined by Silberschatz et al. as "a way to store and retrieve database information that is both convenient and efficient." (2011, 1) and in practice are designed "to manage large systems of information" (2011, 1). For this project, although a large volume of data will not be used, the data that is used is "a collection of interrelated data" (2011, 1) as the data will all contain information related to film shots. Breitinger et al. notes that the efficiency of database lookup is "O(x) (or O(N)) where x is the number of digests in a database" (2014). As the number of items that are planned to be used is relatively small, the overall performance of this project should not be adversely affected. Thus another requirement must be to:

- Design a database like system to create the outputted shot list

Chapter 4

Design & Methodology

4.1 Project Plan

Over the course of this project, the aims and objectives were continually adapting and changing as the needs and requirements of users were further understood. The initial proposal of this project was the following (Byrne, 2022):

4.1.1 Initial Aim

- To develop an effective user experience utilizing a tool that enables laypeople to create cinematic content for games

4.1.2 Initial Objectives

- Evaluate by 21st November two pre-existing software that exist for games and outline what features to incorporate in this software
- Evaluate by 29th November two pre-existing software that exist outside of games and outline what features to incorporate in this software
- Define a minimal set of features required for cinematic content from those surveyed by 7th December.
- Design a user interface via paper prototyping that enables a layperson to create cinematic content by 15th December.
- Collect pre-existing assets online that can be used for sets, music and characters to replicate a genre of film, such as a Western by 30th December.
- Create a minimal viable software that can be used by participants to replicate cinematography for cinematics by 1st March.
- Evaluate if each task is completed in the timeframe outlined for each sprint and create strategies if unsuccessful by 9th March.

- Decide on a similar software that participants can use to make a comparison to when using this software by 17th March.
- Evaluate the design by getting users to attempt to make an example from an existing movie scene using the toolset by 8th April.

Equally a Gantt Chart and table were also created to help manage these objectives, outlined in Figure 3.

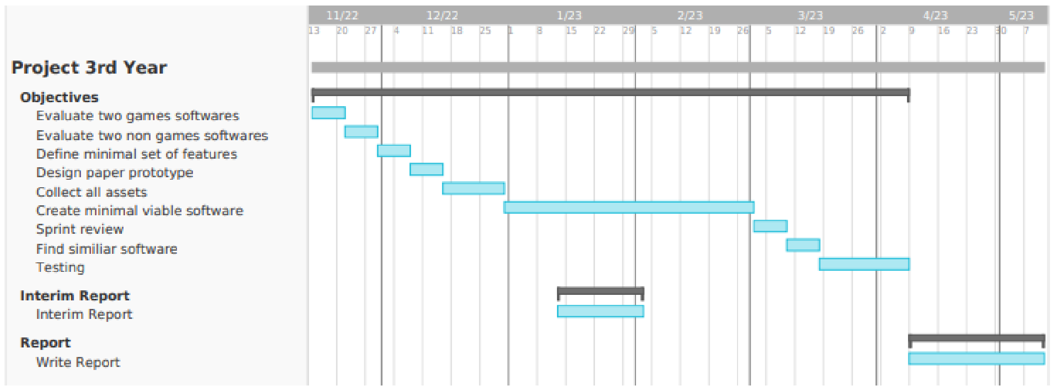


Figure 3: A figure showing the initial version of the Gantt Chart of this project (Byrne, 2022)

4.1.3 The Changing Project Plan

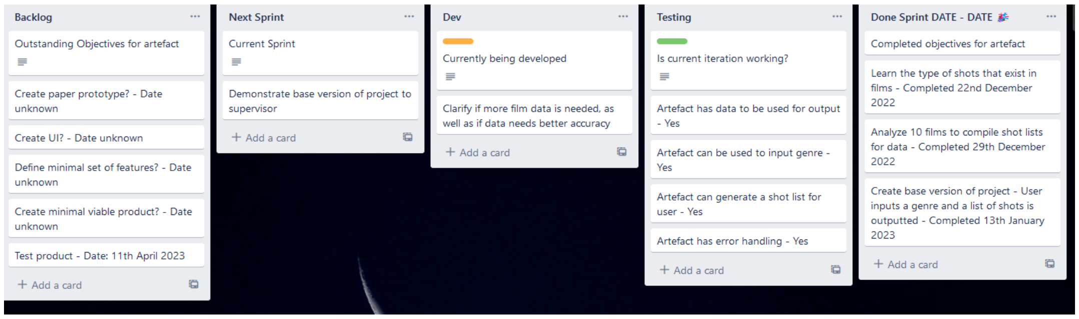
As can be seen in Figure 3, the initial plan for this project was to use the agile methodology SCRUM. The reason for using this methodology was because “agile is best suited for quick and effective development of software” (Srivastava et al, 2017, 1) and in general “is currently the top most technique used in development not only for software but even in the fields of finance” (Srivastava et al, 2017, 1) suggesting that as a technique it has proven to be reliable in software development. In fact, when comparing with a software methodology such as waterfall, it was a more beneficial methodology. An agile methodology such as SCRUM “has a high probability of success” (Mahalakshmi et al, 2013, 4) compared to waterfall whilst also being able to “expect changes and accept the changes” (Mahalakshmi et al, 2013, 4) which waterfall is not able to accommodate. As it was likely that changes would be needed during the development of the artefact, SCRUM seemed a more suitable choice compared to the more limited waterfall method.

Over time, the primary aim of the project and its objectives changed, with the aim now being:

- "Is it possible to create an algorithm that can accurately predict a set of shots a director should employ before creating a film?"

This being the same aim as the finalised aims and objectives. When analysing the games and non-games software it became apparent that they had already fulfilled the criteria set out in the original objectives, that they were simple for the layperson to use to create cinematic content. It was then that it was decided to update the aims and objectives, together with the software methodology. The revised set of objectives are outlined below, with a Scrumban board detailing the progress from the interim report:

- Learn the type of shots that exist in films by 28th December 2022
- Analyse 10 films from which to compile shot lists for data by 5th January 2023
- Complete first iteration by 13th January 2023
- Demonstrate first iteration of project by 3rd February 2023
- Test product by 11th April 2023



Backlog

- Outstanding Objectives for artefact
- Create paper prototype? - Date unknown
- Create UI? - Date unknown
- Define minimal set of features? - Date unknown
- Create minimal viable product? - Date unknown
- Test product - Date: 11th April 2023
- + Add a card

Next Sprint

- Current Sprint
- Demonstrate base version of project to supervisor
- + Add a card

Dev

- Currently being developed
- Clarify if more film data is needed, as well as if data needs better accuracy
- + Add a card

Testing

- Is current iteration working?
- Artefact has data to be used for output - Yes
- Artefact can be used to input genre - Yes
- Artefact can generate a shot list for user - Yes
- Artefact has error handling - Yes
- + Add a card

Done Sprint DATE - DATE

- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- + Add a card

Figure 4: A figure showing the Scrumban board used in this project (Byrne, 2023)

Some objectives shown in Figure 4 were undated as at the interim stage it was not yet clear exactly what features would be required for this project, i.e. a paper prototype and a UI. As all the features were not known it was not possible to predict when a minimal viable software could be completed. However, it was clear that an agile methodology was still required for this project due to the potential for the objectives to continually change. A shift from SCRUM to Scrumban was decided upon using a website called Trello (Atlassian, 2020) to visualise the Scrumban board. Scrumban is a mixture of SCRUM and Kanban. Kanban works by using a Kanban board which "is divided into columns that represent part of the development process" (Petricioli et al, 2022, 2) and is another branch of agile. It is noted that combining Scrum and Kanban "can lead to quicker development, an improved workflow, an improved defect fix rate" (Petricioli et al, 2022, 3) which makes it a more suitable process for this project. It also "keeps iterations, but are usually shorter than sprints" (Petricioli et al, 2022 4). By having shorter development cycles, potential issues can be identified much quicker, meaning any improvements to the software can occur sooner. Therefore, the plan for the Scrumban board was to work on specific tasks taken from the backlog for each sprint and also planning ahead by allocating tasks to be included in the next sprint. There is also a section for testing the current sprint of the project. Any tasks fully completed are placed on the right hand side of the board. This quicker methodology was much more suited to the needs of this project and was therefore adopted, the backlog being adapted as further improvements in each iteration were identified. Further iterations to the Trello board are shown in Appendix A.

4.2 Risk Analysis

A table showing the risk analysis for the project is displayed in Table 2.

Table 2: The Risk Analysis Table

Risk	**Explanation**	**Impact**	**Likelihood**	**Mitigation**
Incorrect identification of film shots	As mentioned in the requirement analysis, a database like system is implemented for this project, however the data used by the system is from human analysis, which is prone to human error	High	High	More than one human should analyse the same film scenes and each film scene should be checked more than once to ensure no shot was identified incorrectly
Insufficient number of films analysed	Since the shot list outputs the most common shot from each part of the sequence, a small volume of films will lead to a more arbitrary choice as there is less chance	High	Medium	Check each film analysed to establish if there is a common shot at each point. If not, more data

	of a common shot being identified			should be given to the algorithm to provide a more informed choice
Varying shot list lengths	Should the film clips analysed be of differing lengths to one another, this could lead to the algorithm having a shot list where the final section is being generated from only one film	Medium	High	Find the point in which the shortest shot list ends and for every film, only loop through the number of shots equal to the shortest list
Not enough participants	Since this project is testing if the software is a good recommendation system, enough users will be needed to check how effective the software is for this purpose	Low	Medium	Start testing early, as soon as the project is complete and advertise testing well

4.3 Design

4.3.1 Front End

Over time, it was decided that a suitable UI was required, further explained in the implementation section. As agile practices were being employed, user experience design could be easily and effectively incorporated into the Scrumban, with Najafi et al. noting in a case study how when user experience was incorporated into each sprint “The results of user research and testing were instrumental in prioritizing the product backlog” (2008, 2). This would mean that it would be most beneficial for the project to perform a final iteration after user testing to address any new items for the product backlog, stemming from user feedback. In the initial aims and objectives, the potential of designing a brand new interface was considered. This was also considered in the second iteration at the interim stage. However, as the back-end of the project had already been designed, the development of a completely new UI was considered inadvisable. Ferreira et al. notes how, in agile projects involving UI, “The projects we studied accepted a considerable amount of UI design up front, but then maintained a connection between the UI design process and development iterations” (2007, 9). As can be seen from the interim, the functionality of the project had already been implemented before the UI design was even proposed, making it more difficult to design UI functionality in conjunction with the back-end development. Instead, it was decided to extend an existing UI from prior game engines studied, such as Unity or Unreal, as they are likely to have pre-existing UI standards in place which could be incorporated into the project. Unity was chosen, firstly due to it having an established user base and secondly there being more transparent documentation available on how to adapt its editor framework.

4.3.2 Back End

As mentioned in the requirement analysis, a database like structure was used in the back end of development, which executes the following tasks in order: Take user input from the front end of the project, loop through database to find films that match the genre, loop through these matching films to find the most common shot at each point in the sequence, create a new shot list that has the most common shot from each sequence, output this list to a json file to be loaded by the front end. The dataset will be of film scenes containing shoot out scenes and will be selected and analysed by a human researcher.

As the front end is not only using Unity's editor interface but extending it to accept user input and generate a visual output, the back end of Unity's editor API (Unity, undated) is also required, both to load the new shot list from the json file and to load the required images that the user will look through after each generation.

In the next chapter, further detail is given as to how the back end changed as the need for a front end was identified with more refined programming practices being employed as the project developed.

4.4 Toolsets and Machine Environments

The tools used in this project were the following: Unity and Visual Studio 2022 using C#. The version control software was GitHub (specifically using GitHub desktop). The reasons for using Unity have already been outlined, however the justifications for utilising the other tools are outlined in subsections 4.4.1 and 4.4.2.

4.4.1 Version Control

The version control used was Github, a tool commonly used by software developers (as of 2023 Github has stated it has 100 million users) (TechCrunch, 2023). This means that, like Unity, it is proven to be a trustworthy software. In fact, Github's own website states that brands such as Pinterest and Mercedes-Benz trust this application, further reinforcing its credibility (Github, undated). Github works by creating a repository that exists on an individual's account, which can be accessed anytime via command line using Git Bash, or via the application Github Desktop. Any changes to the codebase can be published to the repository and stored online. The desktop application was chosen over the command line for the sake of convenience. In general, especially with novice programmers, "the features in standard command line environments are not as assistive to programmers as visual environments" (Dillon et al., 2012, 1) due to command line being less descriptive. Therefore, to avoid any potential confusion Github Desktop was favoured. Another positive feature of Github is the ability to view prior commit histories, allowing for potential errors to be retrospectively fixed by loading a prior commit of the project if needed.

It should be noted that, while Github is a well-established software for version control, there are other tools available for this purpose. Microsoft has infinite cloud storage available on its OneDrive tool, meaning that multiple versions of this project could be uploaded to OneDrive with no need to worry about a lack of storage. This, however, is not a use of version control, rather it is a means to store multiple versions of a project which is not as fluid as the real time updating that Github achieves. Also, should any errors occur in the project, it is harder to pinpoint where in the development pipeline the error first appears. As Github has greater transparency by allowing the developer to look through the relevant files in the repository for each commit, Github Desktop is a far more reliable solution than simple cloud storage and, due to its popularity, there was no real reason to look for other version control software.

4.4.2 Visual Studio C#

For the purpose of back end development, a database like structure was used to store film data. There are many program languages available to do this, one of them being SQL. SQL stands for Structured Query Language and is commonly used for scalable database systems that have tables “with one or more named columns, each having a data type” (Melton, 1996, 1) and “rows having columns corresponding to the table’s columns”. (Melton, 1996, 1). As it is a query based language, data in the database can be easily searched for using the keyword SELECT and specifying a requirement, such as selecting all rows with the first name John. It can also be embedded using other languages, with applications such as C, COBOL and Java (Melton, 1996, 2) (Van Den Brink et al. 2007, 8). However, as mentioned, SQL works firstly with multiple tables that normally handle a multitude of data. An example of a practical use of databases is described by Connolly et al. “The personal details that you supply, such as name, address, age and whether you drink or smoke, are used by the database system to determine the cost of insurance” (2015, 54). There is not much data this project is required to search through, when analysing films all that is required is the genre the film falls under and a list of shots related to the film. This only requires the use of two columns. The use of multiple tables and multiple columns is therefore superfluous to the requirements of this project and the use of SQL would likely not be utilised effectively.

Instead, C# was used as the shot lists could be stored as an array/list and the inbuilt API LINQ could instead be used to look through the lists themselves. LINQ uses the language of SQL such as SELECT but performs the operations instead on C# structures like an array or list, which is the exact task required for the back end.

This also fulfils the database like requirement for the back-end as each list would act as its own separate row, with LINQ acting as the query portion of SQL to get the required information in each row. As stated earlier, Unity was decided as the front end portion of the program later on in development. As Unity also happens to use C# for game development, this influenced the decision to use Unity as well.

4.5 Testing

Fundamentally, two things are being tested in this project. Firstly, the aim “Is it possible to create an algorithm that can accurately predict a set of shots a director should employ before creating a film?” This will be done via user testing. The plan for each user is the following:

- Check each user’s prior knowledge of film shots, if they need a brief refresher, show a quick video that explains each type of shot
- Ask each user to compile a shot list of either an Action, Comedy, Thriller or Western shoot out scene
- Ask each user to compile the outputted shot list from the algorithm
- Ask the user via a questionnaire whether or not they agree with the output generated

As the plan for this software is to be a recommendation system to suggest better shots to users than the ones they had originally thought of, it is important to establish how effective the algorithm is at predicting new shot lists for films. If the results indicate that the system is either not recommending shots the user finds useful or not producing a shot list the user finds of equal or better quality than their own, then it is clear that the project needs further refining in order to better fulfil the aim.

A secondary consideration is that, both as a system that is incorporating Unity's UI and as a recommendation system, the project must also conform to user experience design principles. Again, the most effective way of checking this is via a questionnaire, specifically using likert scales that measure between 1-5, 1 being poor and 5 being exemplary. There may also be further information to be obtained about the system by asking open ended questions, which rely more on a user's personal opinions. The main reason for requesting this type of feedback is that they are "avoiding the bias that may result from suggesting responses to individuals" (Reja et al., 2003, 3). This form of feedback is much less restrictive and may potentially reveal details relating to the system that had not been identified by the questionnaire. Closed end questions that require likert scale responses are still used as there is a potential with purely open ended questionnaires of "larger item non-response" (Reja et al., 2003, 3) which would be hugely detrimental to this project as it requires user feedback. It is generally advised to use a mix of both closed and open ended questions. For instance, "open-ended questions can be used to explore deviant responses to the close-ended questions" (Reja et al., 2003, 4). Taking all this into consideration, the questionnaire used will start with close ended questions finishing with one open ended question allowing users to explain why they picked the options for their likert scales. While this final question may not have as many responses, the responses given could identify issues that should be taken into account when deciding how the project could be further improved going forward.

Appendix B shows the final questionnaire used in testing. Participants will be recruited by asking potential testers if they are interested in the project and its aims. Ethical procedures will be followed by having completely anonymous questionnaires that are only held by the researchers as well as ensuring that consent is obtained from the user and that they have read the participant information sheet. Once all participants have been sourced and all questionnaires analysed, data will be structured by identifying the modal number in each likert scale, indicating what each user found the most/least adequate for each faculty of user experience. Any noteworthy open ended responses will also be recorded.

Chapter 5

Implementation

As Pries explains, SCRUM is an iterative process (1) which means that for every iteration it is possible for "the customer to add changes to the product" (Pries, 2010, 1) with the customer prior to user testing being the developer. This is the case for any agile method, including Scrumban, as they are all iterative processes. The following section outlines each version of the software and the reasons why a new iteration was needed.

5.1 Version 1

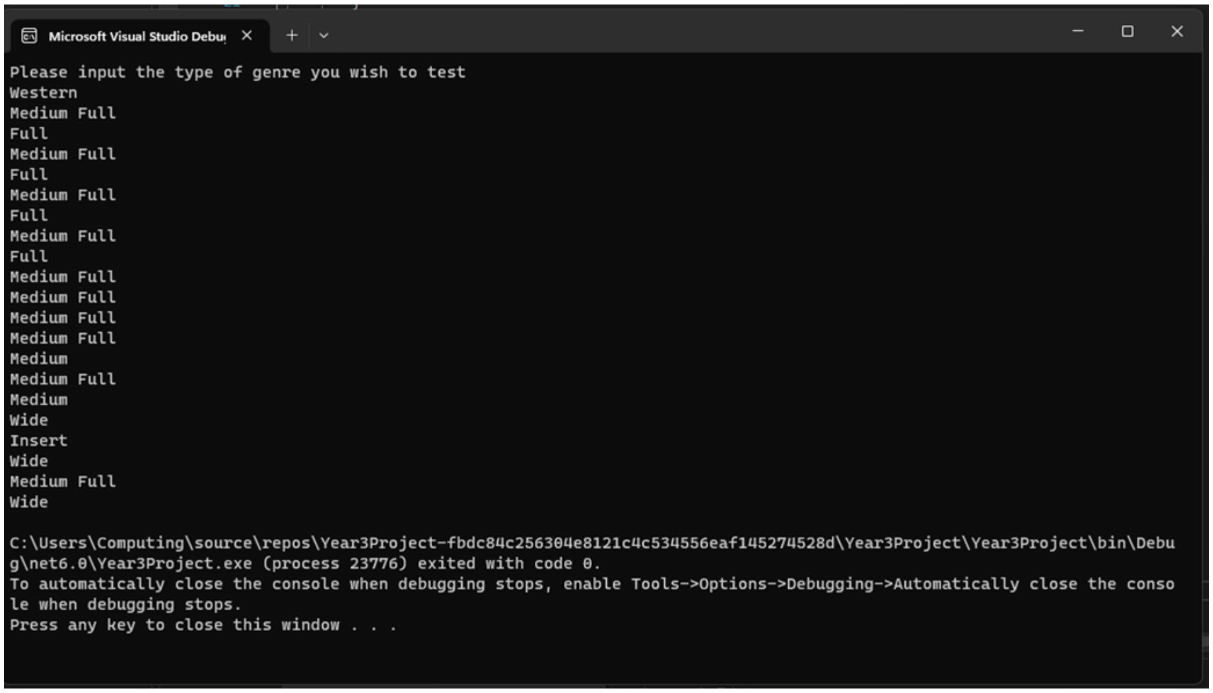
Prior to implementation, there was a stated objective to analyse ten separate films and compile shot lists that could be used as input data for the algorithm. Each film analysed had the common theme of a shoot out scene. Table 3 shows each film chosen, the genre of each film and the number of shots used.

Table 3: A table outlining the ten films analysed for the system

Film	**Genre**	**No of Shots**	**Citation**
The Good, The Bad and The Ugly	Western	46	(Leone, 1966)
Zulu	Action	34	(Endfield, 1964)
Memento	Thriller	17	(Nolan, 2000)
John Wick	Action	36	(Stahelski, 2014)
Grosse Point Blank	Comedy	46	(Armitage, 1997)

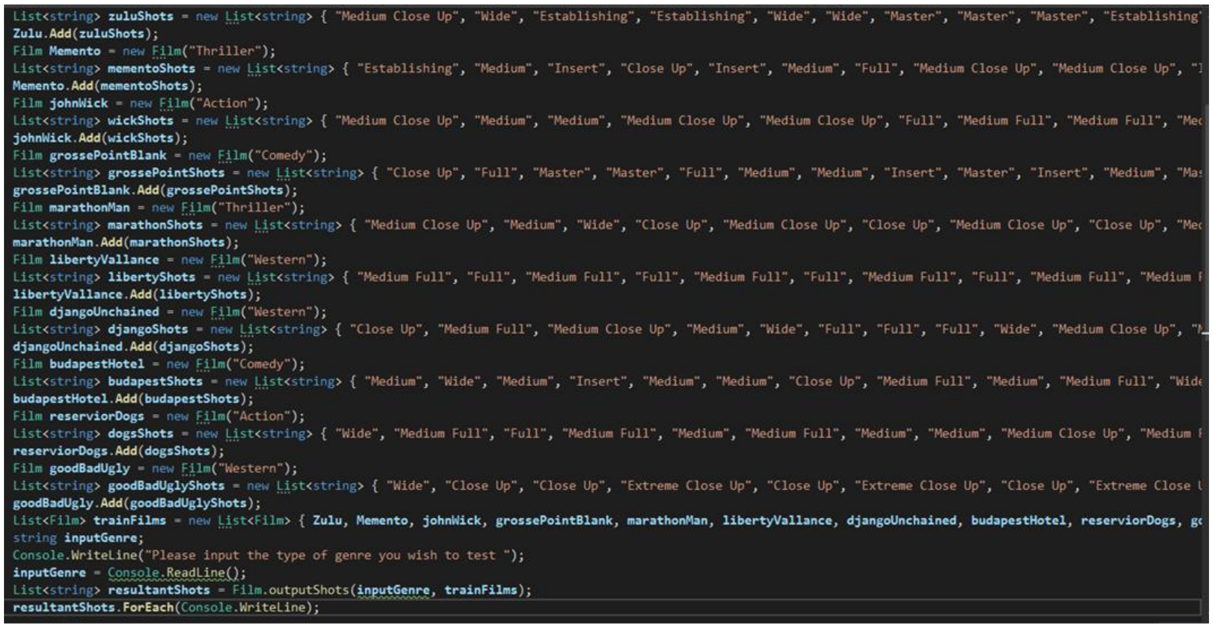
The Grand Budapest Hotel	Comedy	18	(Anderson, 2014)
Marathon Man	Thriller	34	(Schlesinger, 1976)
The Man Who Shot Liberty Valance	Western	20	(Ford, 1962)
Reservoir Dogs	Action	24	(Tarantino, 1992)
Django Unchained	Western	21	(Tarantino, 2012)

In the first version of this project, a simple console application was produced which did the following. In the main function, arrays of films were created that hardcoded the shot lists for each film. Each film was then appended to a further list that would be used to loop through each film that matched the user specified genre. Another for loop would be used to iterate through the films of the specified genre, find the most common shot at each index, before then outputting each common shot to the console. These would therefore be considered the shots that should be used when a user creates their own film, as the more frequently a shot is found at this position, then the more favoured it is to be used by established filmmakers.



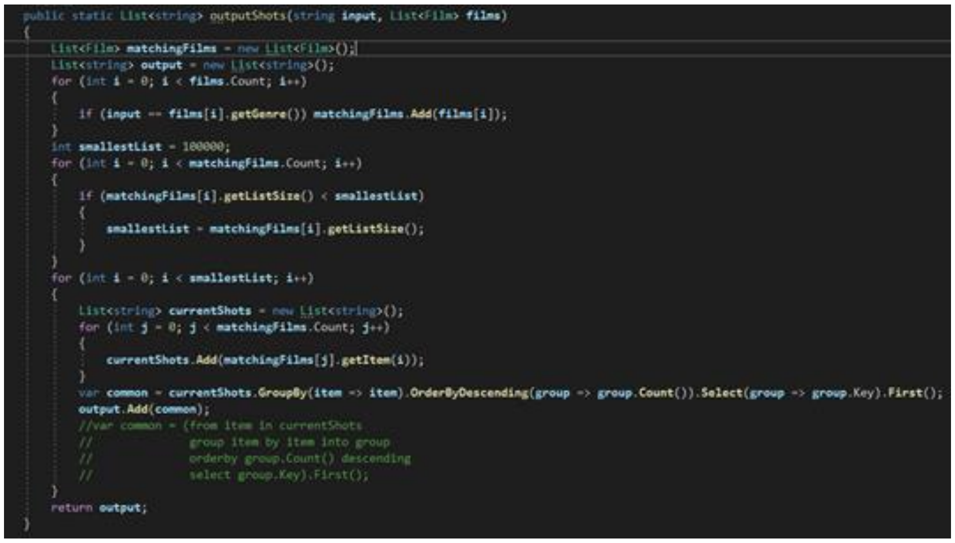
```
Please input the type of genre you wish to test
Western
Medium Full
Full
Medium Full
Full
Medium Full
Full
Medium Full
Full
Medium Full
Medium Full
Medium Full
Medium Full
Medium
Medium Full
Medium
Wide
Insert
Wide
Medium Full
Wide

C:\Users\Computing\source\repos\Year3Project-fbdc84c256304e8121c4c534556eaf145274528d\Year3Project\Year3Project\bin\Debu
g\net6.0\Year3Project.exe (process 23776) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figure 5: The resulting shot list from Version 1



```
List<string> zuluShots = new List<string> { "Medium Close Up", "Wide", "Establishing", "Establishing", "Wide", "Wide", "Master", "Master", "Master", "Establishing"
Zulu.Add(zuluShots);
Film Memento = new Film("Thriller");
List<string> mementoShots = new List<string> { "Establishing", "Medium", "Insert", "Close Up", "Insert", "Medium", "Full", "Medium Close Up", "Medium Close Up", "I
Memento.Add(mementoShots);
Film johnWick = new Film("Action");
List<string> wickShots = new List<string> { "Medium Close Up", "Medium", "Medium", "Medium Close Up", "Medium Close Up", "Full", "Medium Full", "Medium Full", "Med
johnWick.Add(wickShots);
Film grossePointBlank = new Film("Comedy");
List<string> grossePointShots = new List<string> { "Close Up", "Full", "Master", "Master", "Full", "Medium", "Medium", "Insert", "Master", "Insert", "Medium", "Mas
grossePointBlank.Add(grossePointShots);
Film marathonMan = new Film("Thriller");
List<string> marathonShots = new List<string> { "Medium Close Up", "Medium", "Wide", "Close Up", "Medium Close Up", "Close Up", "Medium Close Up", "Close Up", "Med
marathonMan.Add(marathonShots);
Film libertyVallance = new Film("Western");
List<string> libertyShots = new List<string> { "Medium Full", "Full", "Medium Full", "Full", "Medium Full", "Full", "Medium Full", "Full", "Medium Full", "Medium F
libertyVallance.Add(libertyShots);
Film djangoUnchained = new Film("Western");
List<string> djangoShots = new List<string> { "Close Up", "Medium Full", "Medium Close Up", "Medium", "Wide", "Full", "Full", "Full", "Wide", "Medium Close Up", "M
djangoUnchained.Add(djangoShots);
Film budapestHotel = new Film("Comedy");
List<string> budapestShots = new List<string> { "Medium", "Wide", "Medium", "Insert", "Medium", "Medium", "Close Up", "Medium Full", "Medium", "Medium Full", "Wide
budapestHotel.Add(budapestShots);
Film reserviorDogs = new Film("Action");
List<string> dogsShots = new List<string> { "Wide", "Medium Full", "Full", "Medium Full", "Medium", "Medium Full", "Medium", "Medium", "Medium Close Up", "Medium F
reserviorDogs.Add(dogsShots);
Film goodBadUgly = new Film("Western");
List<string> goodBadUglyShots = new List<string> { "Wide", "Close Up", "Close Up", "Extreme Close Up", "Close Up", "Extreme Close Up", "Close Up", "Extreme Close U
goodBadUgly.Add(goodBadUglyShots);
List<Film> trainFilms = new List<Film> { Zulu, Memento, johnWick, grossePointBlank, marathonMan, libertyVallance, djangoUnchained, budapestHotel, reserviorDogs, go
string inputGenre;
Console.WriteLine("Please input the type of genre you wish to test ");
inputGenre = Console.ReadLine();
List<string> resultantShots = Film.outputShots(inputGenre, trainFilms);
resultantShots.ForEach(Console.WriteLine);
```

Figure 6: The hardcoded shot lists



```
public static List<string> outputShots(string input, List<Film> films)
{
    List<Film> matchingFilms = new List<Film>();
    List<string> output = new List<string>();
    for (int i = 0; i < films.Count; i++)
    {
        if (input == films[i].getGenre()) matchingFilms.Add(films[i]);
    }
    int smallestList = 100000;
    for (int i = 0; i < matchingFilms.Count; i++)
    {
        if (matchingFilms[i].getListSize() < smallestList)
        {
            smallestList = matchingFilms[i].getListSize();
        }
    }
    for (int i = 0; i < smallestList; i++)
    {
        List<string> currentShots = new List<string>();
        for (int j = 0; j < matchingFilms.Count; j++)
        {
            currentShots.Add(matchingFilms[j].getItem(i));
        }
        var common = currentShots.GroupBy(item => item).OrderByDescending(group => group.Count()).Select(group => group.Key).First();
        output.Add(common);
        //var common = (from item in currentShots
        //              group item by item into group
        //              orderby group.Count() descending
        //              select group.Key).First();
    }
    return output;
}
```

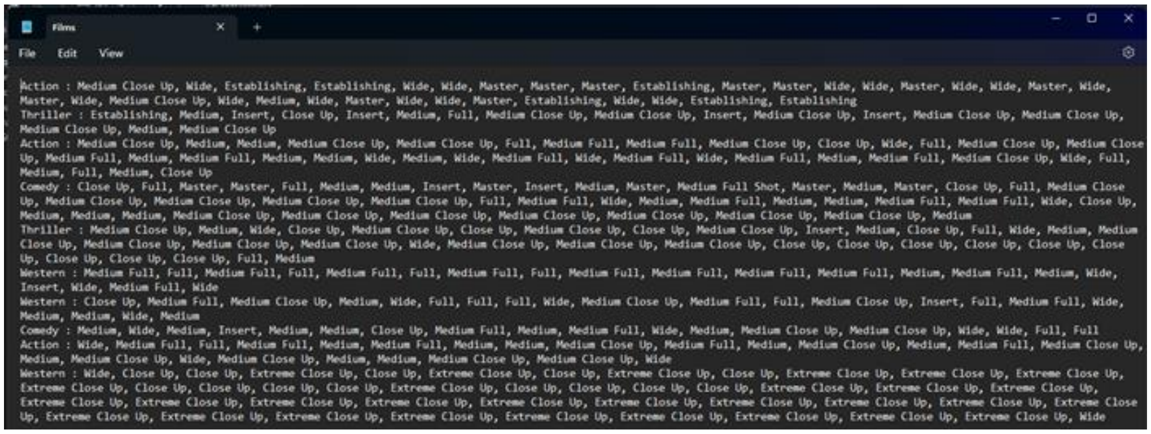
Figure 7: A figure showing an overview of Version 1 of the project.

As can be seen in Figure 7, there is a main function and a class called Film which will be used to create both film objects with their relevant shot lists and output the final shot list to the user. LINQ is used as it will order shots in descending order via the use of a GroupBy method, following this tutorial (StackOverflow, 2008). For reasons outlined in Chapter 4, LINQ has been used. It should be noted that Table 3 shows films with differing numbers of shots. As the aim is to compare the most common shots at each position, the shot list was decided to be only the length of the shortest film in the genre, to prevent a scenario where any later shots proposed would only be informed by a single film.

5.2 Version 2

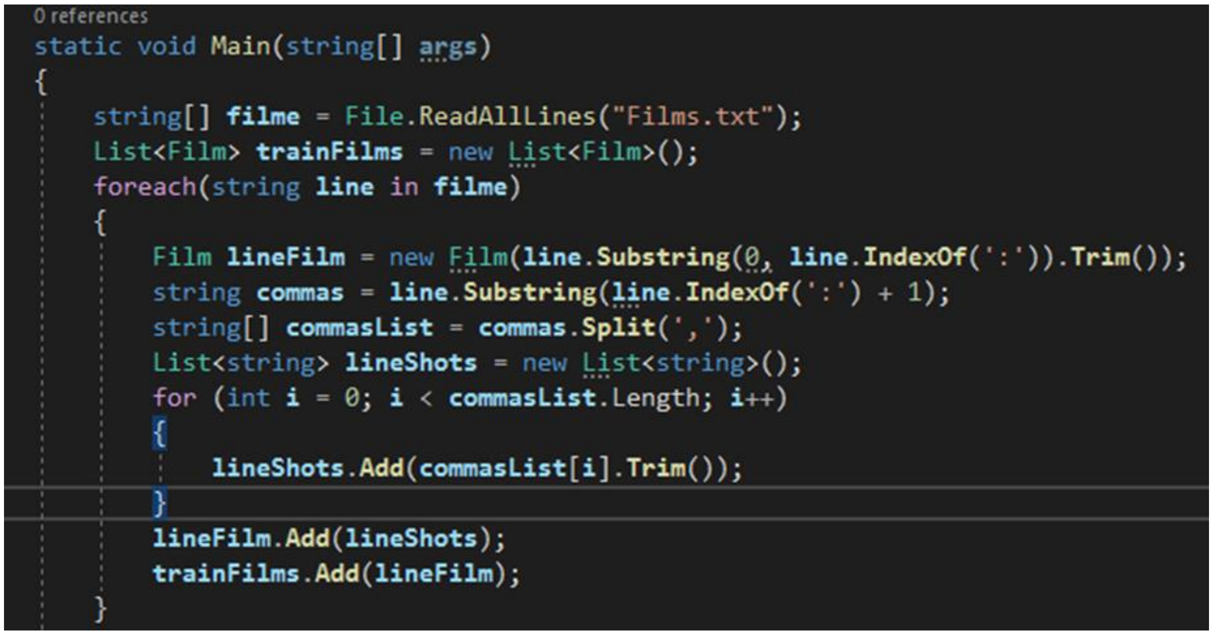
Further on in development it was decided that a second version of this implementation should be included to add more detail as to why each shot was selected by the algorithm. This can be seen as giving system transparency, explained by Zhao et al. as “the extent to which information of a system’s reasoning is provided and made available to users” (2019). The reason that this may be important for this project is that, as the system being created is a recommendation system, it can help laypeople make a more informed choice. As Arnold et al. discovers “the availability of a fully functional explanation facility influences both novices' and experts' judgements" (2006, 94). Showing how the algorithm arrived at each choice may help users understand why each shot selected was recommended to be the best shot to use for each section of the sequence. It could also provide users with alternate shots that they may wish to explore instead, allowing for the benefits of creative freedom as outlined by Resnick.

This second version also cleaned up aspects of the first version, i.e. the hardcoded arrays in the main file were replaced with a txt file that contained each array of shots, together with the specific genre related to these shots. The rationale for doing this is that it was easier to add to the txt file than to hardcode each value into the program. It also helps to reduce the amount of code needed for each film, as all films can be handled and added to the system by one line as opposed to every film needing three lines of code to be instantiated and added to the program.



```
Action : Medium Close Up, Wide, Establishing, Establishing, Wide, Wide, Master, Master, Master, Establishing, Master, Master, Wide, Wide, Master, Wide, Wide, Master, Wide,
Master, Wide, Medium Close Up, Wide, Medium, Wide, Master, Wide, Wide, Master, Establishing, Wide, Wide, Establishing, Establishing
Thriller : Establishing, Medium, Insert, Close Up, Insert, Medium, Full, Medium Close Up, Medium Close Up, Insert, Medium Close Up, Insert, Medium Close Up, Medium Close Up,
Medium Close Up, Medium, Medium Close Up
Action : Medium Close Up, Medium, Medium, Medium Close Up, Medium Close Up, Full, Medium Full, Medium Full, Medium Full, Medium Close Up, Close Up, Wide, Full, Medium Close Up, Medium Close
Up, Medium Full, Medium, Medium Full, Medium, Medium, Wide, Medium, Wide, Medium Full, Wide, Medium Full, Wide, Medium Full, Medium, Medium Full, Medium Close Up, Wide, Full,
Medium, Full, Medium, Close Up
Comedy : Close Up, Full, Master, Master, Full, Medium, Medium, Insert, Master, Insert, Medium, Master, Medium Full Shot, Master, Medium, Master, Close Up, Full, Medium Close
Up, Medium Close Up, Medium Close Up, Medium Close Up, Medium Close Up, Full, Medium Full, Wide, Medium, Medium Full, Medium, Medium, Medium Full, Medium Full, Wide, Close Up,
Medium, Medium, Medium, Medium Close Up, Medium Close Up, Medium Close Up, Medium Close Up, Medium Close Up, Medium Close Up, Medium Close Up, Medium
Thriller : Medium Close Up, Medium, Wide, Close Up, Medium Close Up, Close Up, Medium Close Up, Close Up, Medium Close Up, Insert, Medium, Close Up, Full, Wide, Medium, Medium
Close Up, Medium Close Up, Medium Close Up, Medium Close Up, Wide, Medium Close Up, Medium Close Up, Medium Close Up, Close Up, Close Up, Close Up, Close Up, Close Up, Close
Up, Close Up, Close Up, Close Up, Full, Medium
Western : Medium Full, Full, Medium Full, Full, Medium Full, Full, Medium Full, Full, Medium Full, Full, Medium Full, Medium Full, Medium Full, Medium Full, Medium, Medium Full, Medium, Wide,
Insert, Wide, Medium Full, Wide
Western : Close Up, Medium Full, Medium Close Up, Medium, Wide, Full, Full, Full, Wide, Medium Close Up, Medium Full, Full, Medium Close Up, Insert, Full, Medium Full, Wide,
Medium, Medium, Wide, Medium
Comedy : Medium, Wide, Medium, Insert, Medium, Medium, Close Up, Medium Full, Medium, Medium Full, Wide, Medium, Medium Close Up, Medium Close Up, Wide, Wide, Full, Full
Action : Wide, Medium Full, Full, Medium Full, Medium, Medium Full, Medium, Medium, Medium Close Up, Medium Full, Medium, Medium Close Up, Medium, Medium Full, Medium Close Up,
Medium, Medium Close Up, Wide, Medium Close Up, Medium, Medium, Medium Close Up, Medium Close Up, Wide
Western : Wide, Close Up, Close Up, Extreme Close Up, Close Up, Extreme Close Up, Close Up, Extreme Close Up, Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up,
Extreme Close Up, Close Up, Close Up, Close Up, Close Up, Extreme Close Up, Close Up, Close Up, Close Up, Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up,
Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close
Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Extreme Close Up, Wide
```

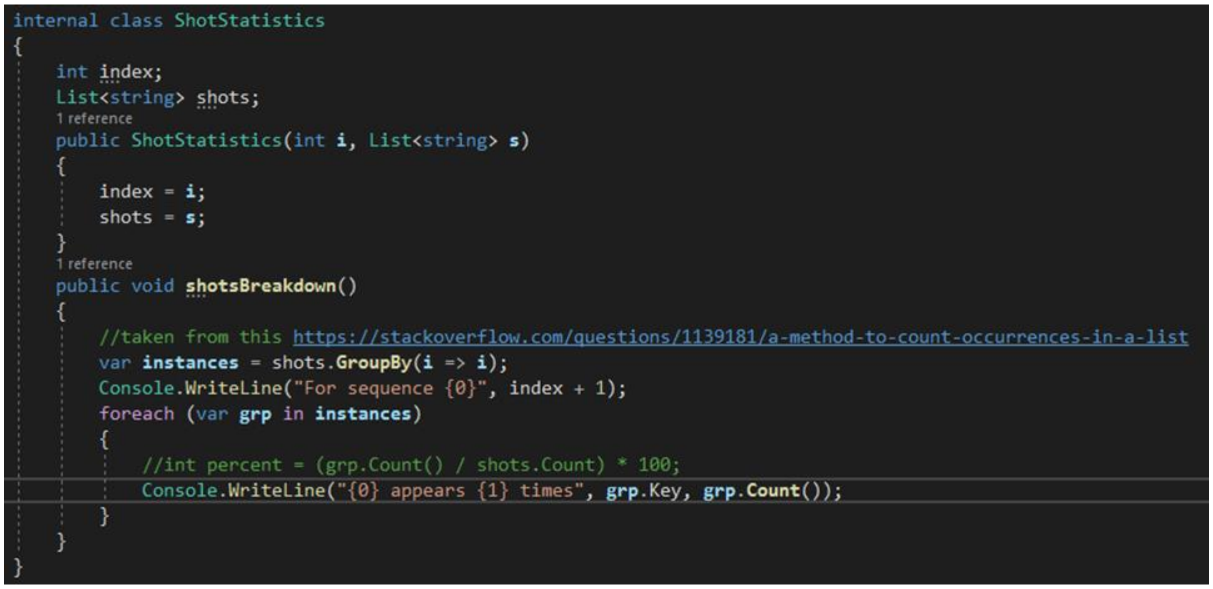
Figure 8: A figure to show the contents of the films.txt file



```
0 references
static void Main(string[] args)
{
    string[] filme = File.ReadAllLines("Films.txt");
    List<Film> trainFilms = new List<Film>();
    foreach(string line in filme)
    {
        Film lineFilm = new Film(line.Substring(0, line.IndexOf(':')).Trim());
        string commas = line.Substring(line.IndexOf(':') + 1);
        string[] commasList = commas.Split(',');
        List<string> lineShots = new List<string>();
        for (int i = 0; i < commasList.Length; i++)
        {
            lineShots.Add(commasList[i].Trim());
        }
        lineFilm.Add(lineShots);
        trainFilms.Add(lineFilm);
    }
```

Figure 9: A figure showing improved code to create film objects.

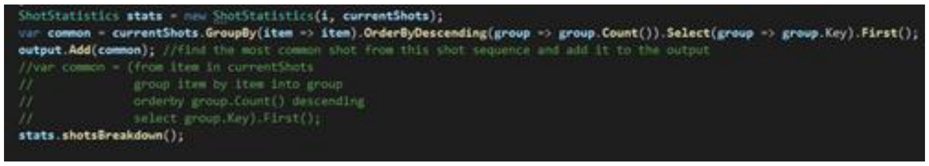
To create a film object, Figure 9 shows that each line of the txt file will be split into a string and an array of strings, with the split happening when the line encounters a colon. These will be variables for each film object. While this does mean that each line of the txt file needs to follow a specific format, the overall process is much more efficient than in the prior version. All other aspects of Version 1 remained the same, however, a new class called ShotStatistics was made in order to incorporate system transparency, created following this forum post (StackOverflow, 2009).



```
internal class ShotStatistics
{
    int index;
    List<string> shots;
    1 reference
    public ShotStatistics(int i, List<string> s)
    {
        index = i;
        shots = s;
    }
    1 reference
    public void shotsBreakdown()
    {
        //taken from this https://stackoverflow.com/questions/1139181/a-method-to-count-occurrences-in-a-list
        var instances = shots.GroupBy(i => i);
        Console.WriteLine("For sequence {0}", index + 1);
        foreach (var grp in instances)
        {
            //int percent = (grp.Count() / shots.Count) * 100;
            Console.WriteLine("{0} appears {1} times", grp.Key, grp.Count());
        }
    }
}
```

Figure 10: A figure to show the new class ShotStatistics

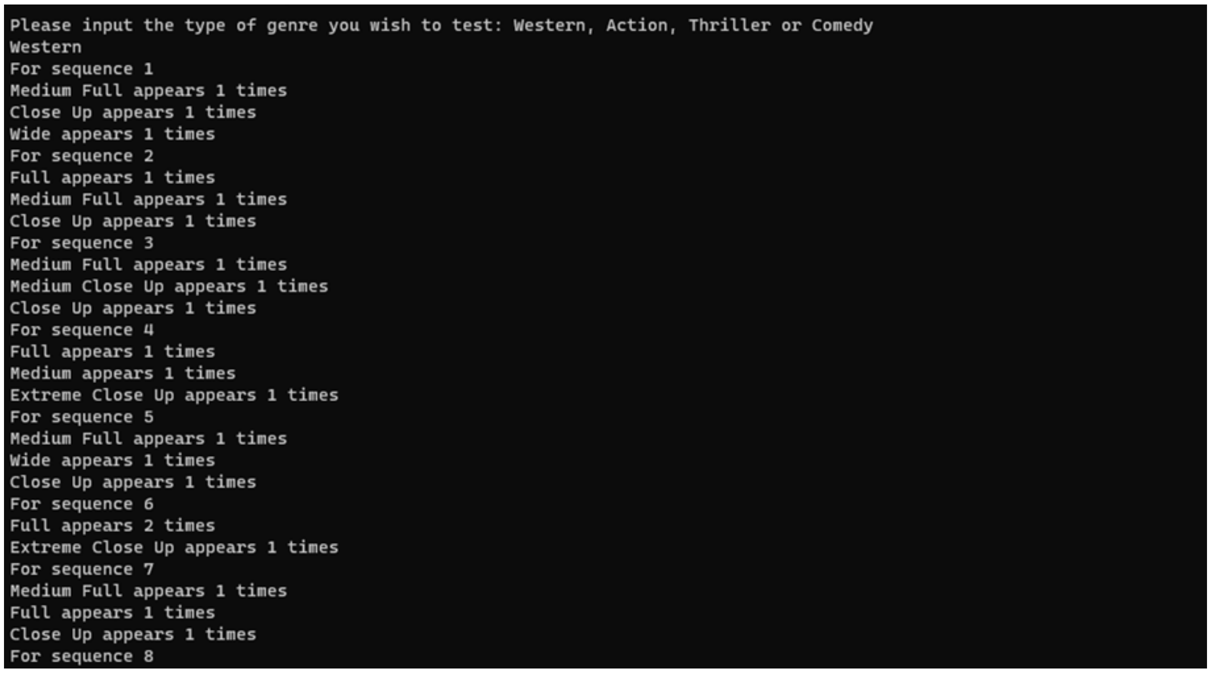
In order to call this class, an extra amendment were made to the function outputShots.



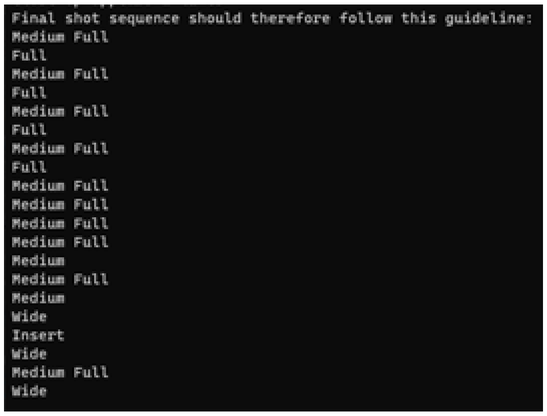
```
ShotStatistics stats = new ShotStatistics(i, currentShots);
var common = currentShots.GroupBy(item => item).OrderByDescending(group => group.Count()).Select(group => group.Key).First();
output.Add(common); //find the most common shot from this shot sequence and add it to the output
//var common = (from item in currentShots
//              group item by item into group
//              orderby group.Count() descending
//              select group.Key).First();
stats.shotsBreakdown();
```

Figure 11: A figure to show how ShotStatistics is called

After making these amendments, the console output was updated to include more information as to why each shot had been recommended to the user.



```
Please input the type of genre you wish to test: Western, Action, Thriller or Comedy
Western
For sequence 1
Medium Full appears 1 times
Close Up appears 1 times
Wide appears 1 times
For sequence 2
Full appears 1 times
Medium Full appears 1 times
Close Up appears 1 times
For sequence 3
Medium Full appears 1 times
Medium Close Up appears 1 times
Close Up appears 1 times
For sequence 4
Full appears 1 times
Medium appears 1 times
Extreme Close Up appears 1 times
For sequence 5
Medium Full appears 1 times
Wide appears 1 times
Close Up appears 1 times
For sequence 6
Full appears 2 times
Extreme Close Up appears 1 times
For sequence 7
Medium Full appears 1 times
Full appears 1 times
Close Up appears 1 times
For sequence 8
```



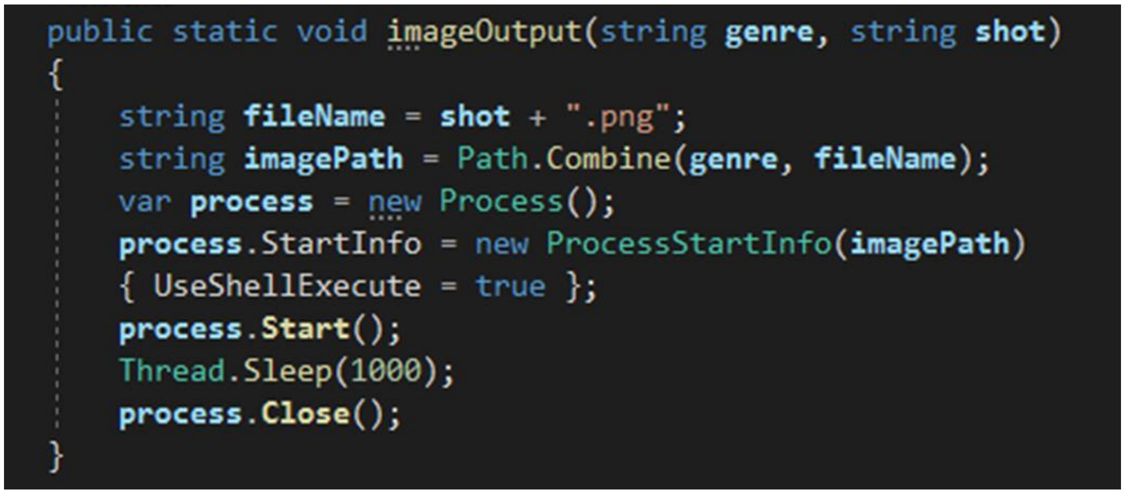
```
Final shot sequence should therefore follow this guideline:
Medium Full
Full
Medium Full
Full
Medium Full
Full
Medium Full
Full
Medium Full
Medium Full
Medium Full
Medium Full
Medium
Medium Full
Medium
Wide
Insert
Wide
Medium Full
Wide
```

Figure 12: A figure showing a more detailed breakdown for the outputted shot list

As can be seen in Figure 12, the user now has a more functional explanation as to why the algorithm has specifically picked each shot for the final shot sequence. It can be seen that for sequence 6 in a western, a full shot appeared twice, whilst an extreme close up appeared only once. Therefore, the algorithm had chosen a full shot to be implemented for the output. In the event of a tie, such as in sequence 1, the algorithm chooses the first element. Whilst the requirement to design a database like system was successfully achieved, this second version still had usability concerns. As Nielson explains, the user needs concepts familiar to them, rather than internal jargon. As the console output only refers to terminology related to film with no real explanation as to what each shot relates to, it fails as a system that can be used by laypeople as it presumes prior knowledge. This, therefore, means that it communicates poorly and a rethink was required in order to better describe the final sequence suggested.

5.3 Version 3

In order to rectify the underlying clarity issue highlighted in the two prior versions it was decided that a visual component to the output was required. Rather than having a simple console interface as a UI, it was hoped this would make the shots more understandable to the user. With visual representations users can “be more concerned with the perception and the meanings attributed to them” (Prosser, 2012, 177). Therefore, a visual representation of the shot list would help users understand the context of each shot and what it would mean in the context of their film. It was believed this would help break down the barriers laypeople may encounter when interpreting the information provided. To achieve this an extra method, called imageOutput, was included.



```
public static void imageOutput(string genre, string shot)
{
    string fileName = shot + ".png";
    string imagePath = Path.Combine(genre, fileName);
    var process = new Process();
    process.StartInfo = new ProcessStartInfo(imagePath)
    { UseShellExecute = true };
    process.Start();
    Thread.Sleep(1000);
    process.Close();
}
```

Figure 13: A figure showing the code for image output

As can be seen in Figure 13, this function in the Film class does the following; in the main function a for loop is used for each shot in the outputted shot list, which then calls this method in Film. A reference to the genre is used as there are folders this program will access that will contain the relevant shots which will then be displayed onto the monitor. A path is made combining the folder name "genre" and the shot i.e. wide. An inbuilt function called Process (Microsoft, undated) is then used, which tells the program to load the image path i.e. Western/Wide.png. Once the process starts, it runs for a second before closing and then the next image is loaded in exactly the same way. An example of how the images are displayed is shown in Figure 14.

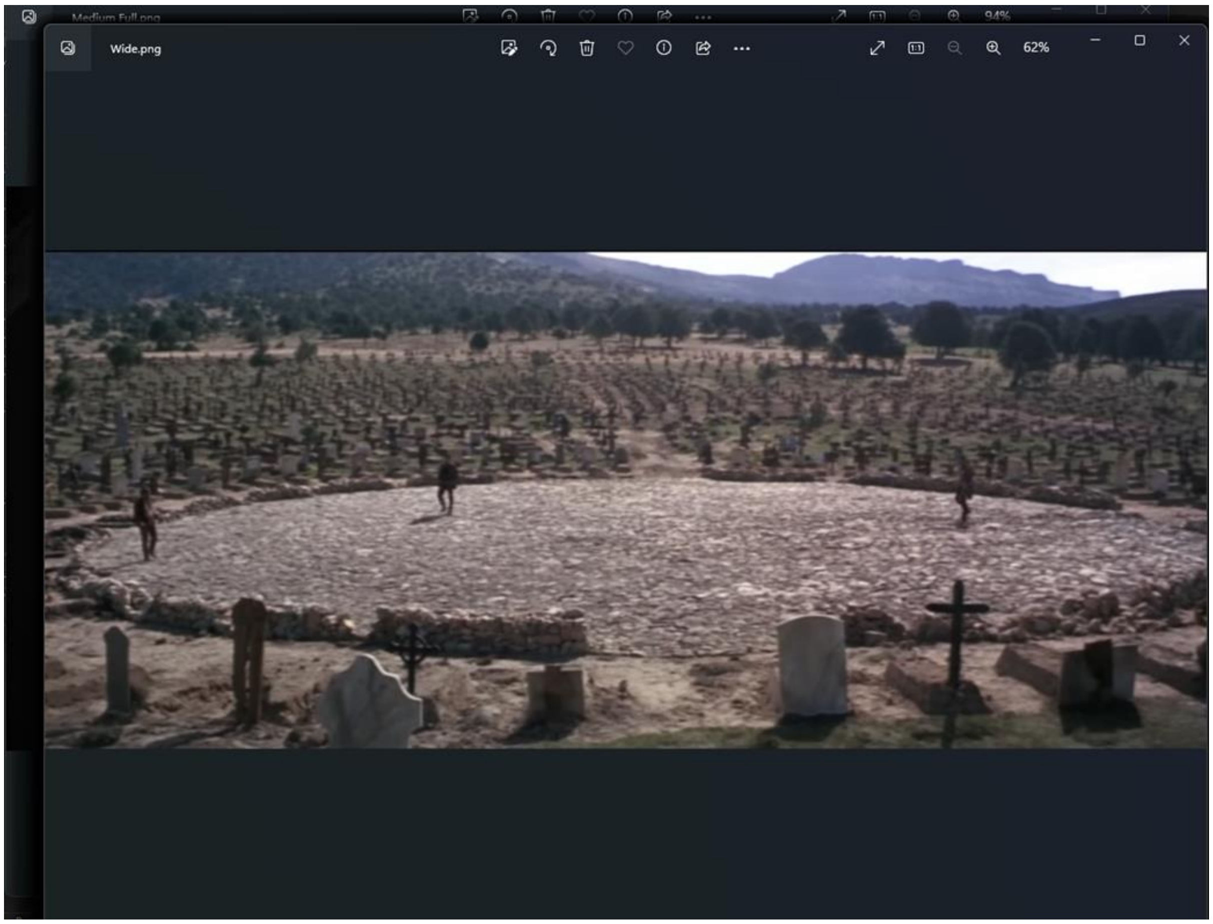


Figure 14: A figure showing how images have been displayed in Version 3 of the project

As can be seen in Figure 14, all images load as separate png images in a sequence that matches the output sequence generated from the algorithm. Shot statistic information and the final outputted shot list can still be viewed on the console window. This now means that the user is better informed as to what each shot means as they have a clear visual representation in line with Prosser's thinking. It also fulfils the requirement to have an output understandable by any user.

5.4 Version 4

It was noted during development of this application that there is a well defined way to implement this program, called MVC (Model View Controller), an important principle in object orientated programming. As Bucanek notes it "describes the architecture of a system of objects." (2009, 353). As C# is designed for object orientated applications a better architecture needed to be employed for the use of this software. An outline of how MVC's normally look is defined in Figure 15.

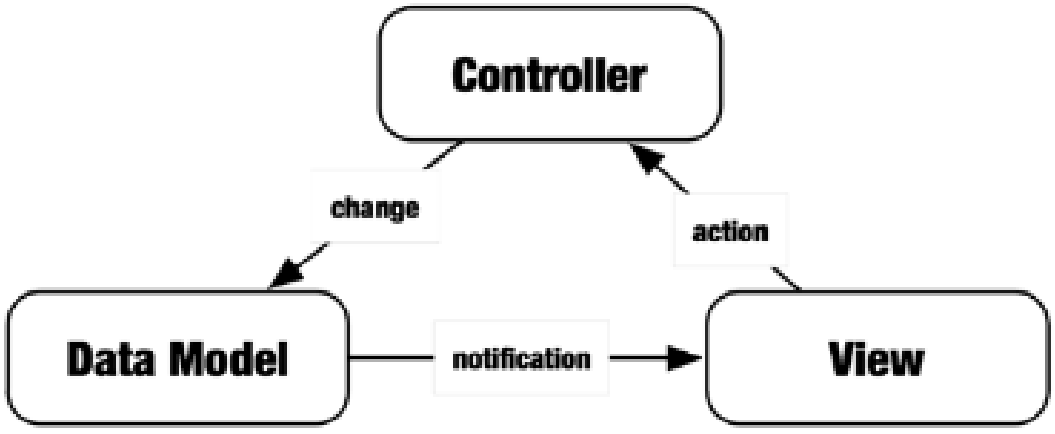
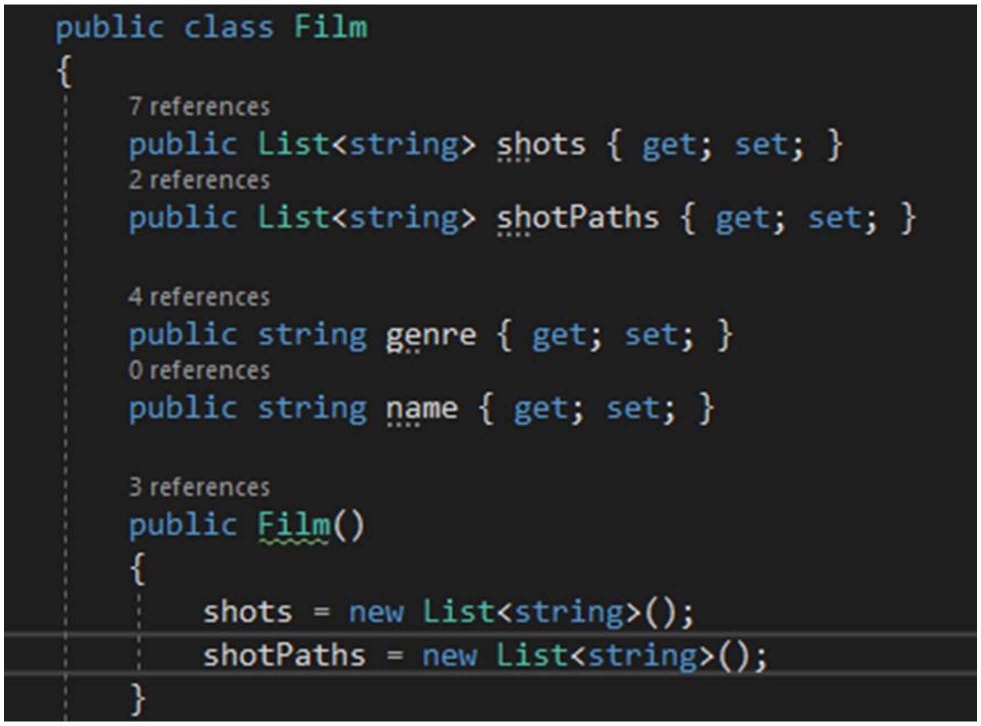


Figure 15: A figure showing the general architecture of an MVC application (Bucanek, 2009, 354)

According to Bucanek, five principles must be adhered to in an MVC application. The Model (or Data Model) must encapsulate information. View must display objects to the user. Controller must implement actions. View must gather user input and pass it to the Controller to perform an action. Finally, the View must observe the Model and update its display when it changes (2009, 354). When looking at the previous versions it is clear the View, Main, violates these principles. Film objects, when created, are handled inside the View rather than in the Controller. There is also no Model, any film objects are also created and stored in the main View class. It was clear, therefore, that an update to the program was needed to stop violating this object oriented principle.

5.4.1 Model

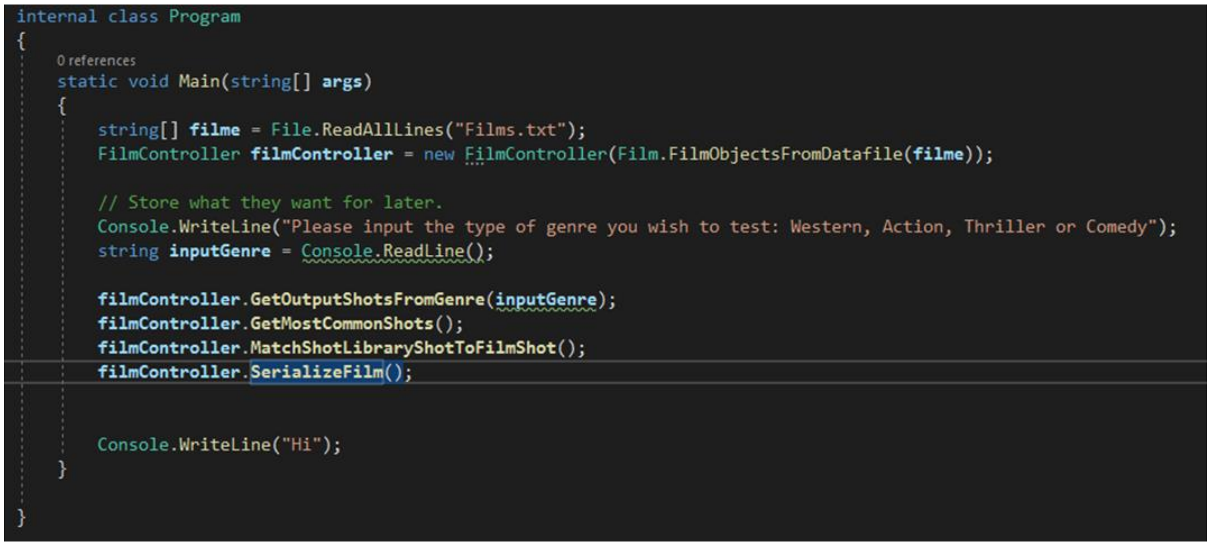


```
public class Film
{
    7 references
    public List<string> shots { get; set; }
    2 references
    public List<string> shotPaths { get; set; }

    4 references
    public string genre { get; set; }
    0 references
    public string name { get; set; }

    3 references
    public Film()
    {
        shots = new List<string>();
        shotPaths = new List<string>();
    }
```

Figure 16: A figure showing the Model of the MVC

As can be seen in Figure 16, the Model is now established. Any information that needs to be encapsulated, such as the list of shots, the shot paths and the genre are now stored in a separate class.

5.4.2 View

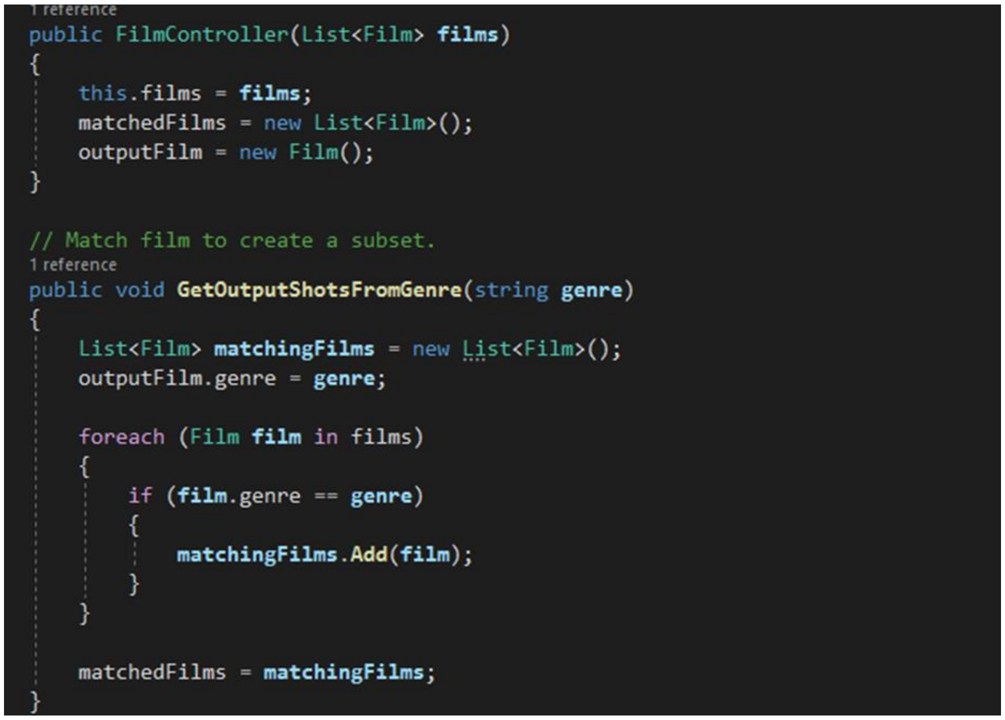


```
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        string[] filme = File.ReadAllLines("Films.txt");
        FilmController filmController = new FilmController(Film.FilmObjectsFromDatafile(filme));

        // Store what they want for later.
        Console.WriteLine("Please input the type of genre you wish to test: Western, Action, Thriller or Comedy");
        string inputGenre = Console.ReadLine();

        filmController.GetOutputShotsFromGenre(inputGenre);
        filmController.GetMostCommonShots();
        filmController.MatchShotLibraryShotToFilmShot();
        filmController.SerializeFilm();


        Console.WriteLine("Hi");
    }

}
```

Figure 17: A figure showing the View part of MVC

As can be seen in Figure 17, the View is now established, handling the user input and passing it to the filmController object for data handling. It can also be seen that the filmController uses a static method in the Model class to generate all the film data that the model needs to encapsulate, meaning the Controller changes the model as outlined in Figure 15.

5.4.3 Controller

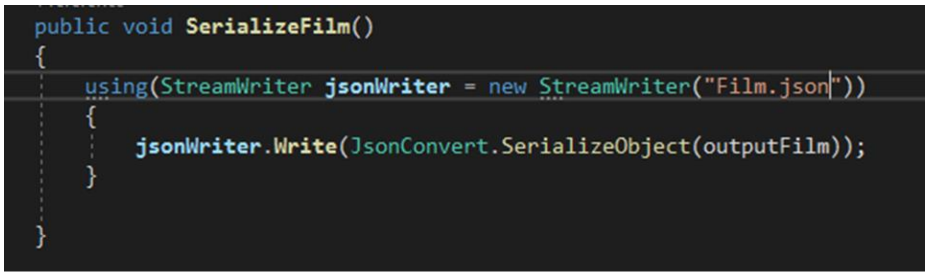


```
1 reference
public FilmController(List<Film> films)
{
    this.films = films;
    matchedFilms = new List<Film>();
    outputFilm = new Film();
}

// Match film to create a subset.
1 reference
public void GetOutputShotsFromGenre(string genre)
{
    List<Film> matchingFilms = new List<Film>();
    outputFilm.genre = genre;

    foreach (Film film in films)
    {
        if (film.genre == genre)
        {
            matchingFilms.Add(film);
        }
    }

    matchedFilms = matchingFilms;
}
```

Figure 18: A figure showing part of the Controller implementation

As can be seen from Figure 18, any of the internal workings of the program, in this case retrieving any films that match the specified genre, are handled purely in the Controller class. Therefore an effective MVC application has been successfully implemented.

One thing to note is that in View a function called SerializeFilm is called in the Controller class. It had become clear that, while displaying images onto the console did help users better understand how film shots are defined, it was also very unprofessional. Multiple images would be loaded onto the monitor, meaning the user would need to close each individual image one by one. Secondly, a general console application was felt to be too simplistic. McKay states that people “react emotionally to a product’s visual appearance” (2013, 9) meaning that if it is of a “questionable quality, users will naturally assume that the rest of the product has the same level of quality" (2013, 9). Again, referring back to Nielson "Users experience with other products set their expectations" (1994). Most users will be more familiar with UIs that have a visual component rather than a simple console application, so in order to create a more professional application it was decided to implement another version using a UI instead.

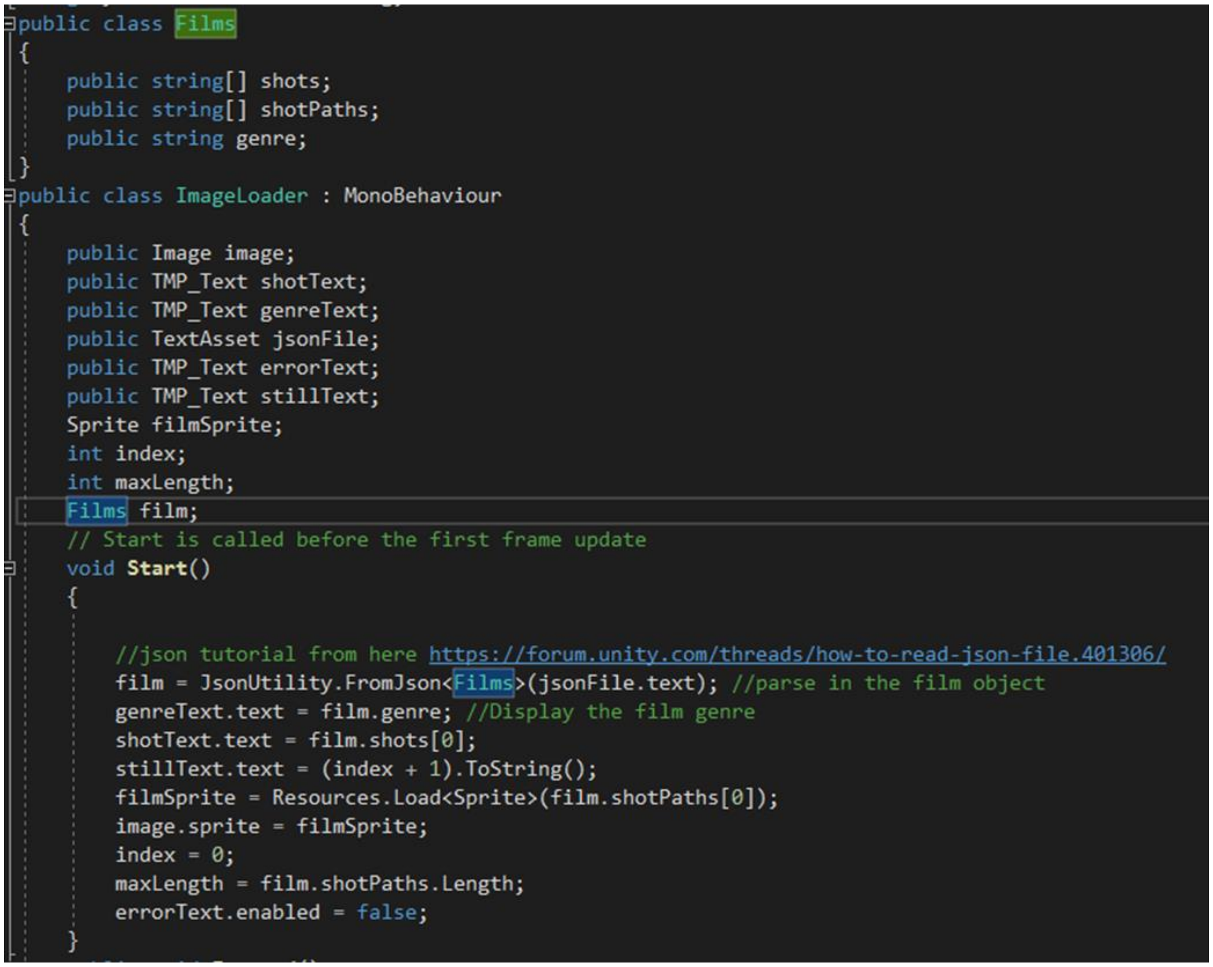


```
public void SerializeFilm()
{
    using(StreamWriter jsonWriter = new StreamWriter("Film.json"))
    {
        jsonWriter.Write(JsonConvert.SerializeObject(outputFilm));
    }

}
```

Figure 19: A figure showing the SerializeFilm function

5.5 Version 5

In the requirement analysis, it was decided that extending Unity's UI to incorporate the functionality of this project would be a logical solution. Initially the Unity project was very simple. Rather than have the images for the final shots displayed as separate png files, it was decided that images would be displayed in Unity's play view by using a canvas game object which, in turn, would have buttons allowing the user to transition easily between different parts of the previs. This was done by loading the json file that was created from the MVC implemented in Version 4 into Unity. This has now become the back end of the project, as shown in Figure 20.



```
public class Films
{
    public string[] shots;
    public string[] shotPaths;
    public string genre;
}
public class ImageLoader : MonoBehaviour
{
    public Image image;
    public TMP_Text shotText;
    public TMP_Text genreText;
    public TextAsset jsonFile;
    public TMP_Text errorText;
    public TMP_Text stillText;
    Sprite filmSprite;
    int index;
    int maxLength;
    Films film;
    // Start is called before the first frame update
    void Start()
    {

        //json tutorial from here https://forum.unity.com/threads/how-to-read-json-file.401306/
        film = JsonUtility.FromJson<Films>(jsonFile.text); //parse in the film object
        genreText.text = film.genre; //Display the film genre
        shotText.text = film.shots[0];
        stillText.text = (index + 1).ToString();
        filmSprite = Resources.Load<Sprite>(film.shotPaths[0]);
        image.sprite = filmSprite;
        index = 0;
        maxLength = film.shotPaths.Length;
        errorText.enabled = false;
    }
```

Figure 20: A figure showing json files implemented in Unity

As can be seen in Figure 20, the json file is loaded. Once loaded, the information is firstly stored into a separate class called Films. This was incorporated from the Unity forums (Unity, 2014). The canvas is then manipulated to load the image at position 0, as well as displaying text for the position in both the shot list and the genre. This is shown in Figure 21.

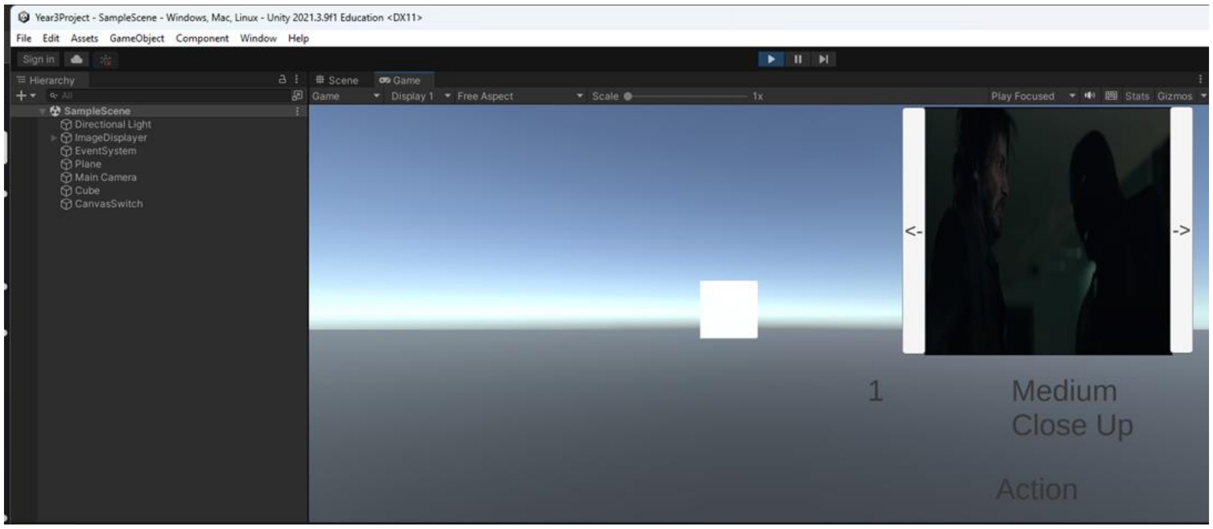
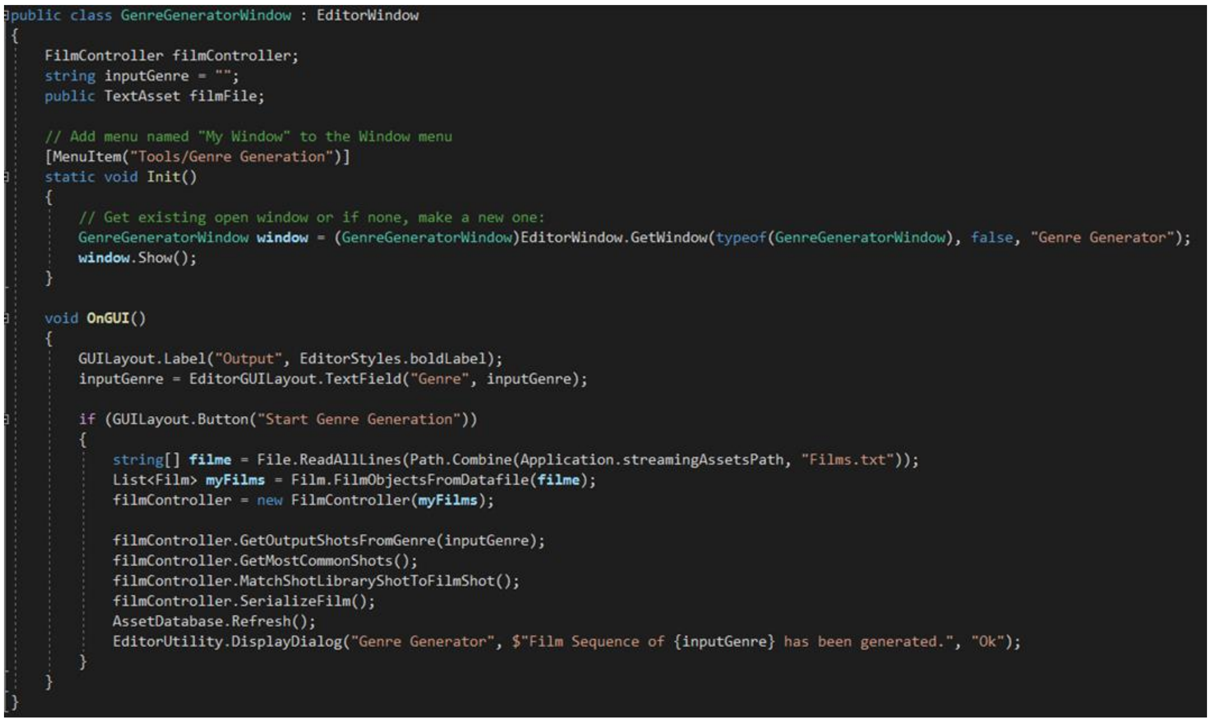


Figure 21: A figure showing the first version of the Unity program

As mentioned by Resnick, it is important to allow users to experiment with the shots in the recommendation system. This means that the user is free to try a different shot than the medium close up selected by the program as illustrated in Figure 21. To enable this, users were given control of Unity's camera in play mode, allowing them to fine tune their sequence to their individual needs, whilst also having the guidance of the software itself to refer back to. This was achieved by following two tutorials (Game Developer Training, 2022) (Unity, 2012). This would therefore fulfil the second requirement of this software, to allow users to experiment with the generated output. However, this version was still not ideal. In order to change the genre, the user would need to exit Unity, load the MVC code implemented in Version 4 for the new genre and then reload the Unity project. This was clearly inefficient and still not professional enough to encourage a user to utilise this software in a practical setting. Also, when users had control of the camera, there was no real point of reference that could be used in order to try and experiment with shots. Although the cube as seen in Figure 21 could be considered a point of reference, it is more likely that directors would film human participants. Therefore, it was not contextually sufficient and would be likely to cause confusion.

5.6 Version 6

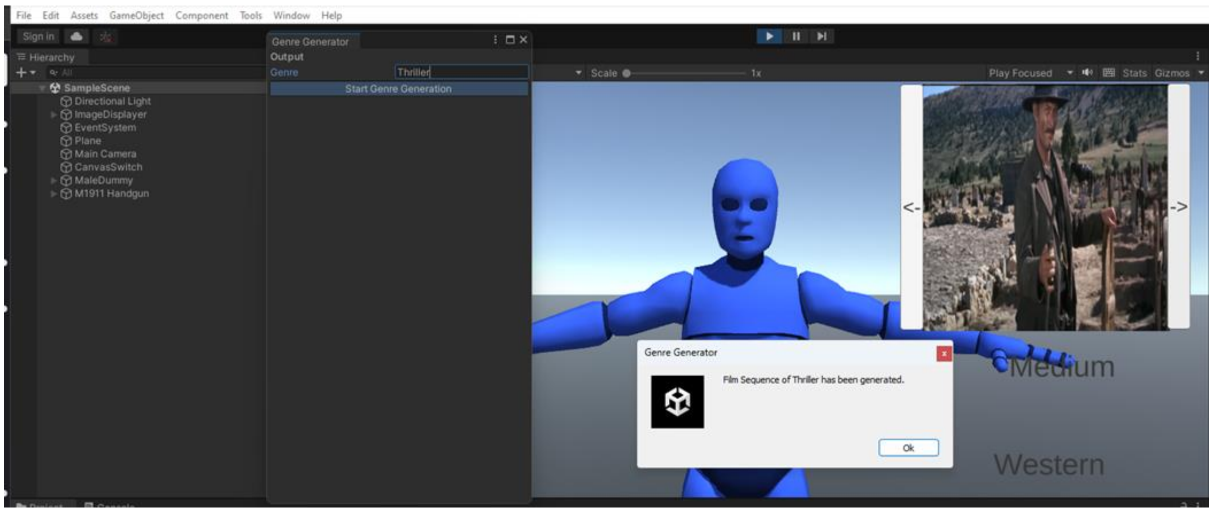
A final version of this project was created before being tested by users. Firstly, the MVC model was incorporated into Unity. Rather than the View class working only for a console application, it was adapted to use the UnityEditor API as a way to handle user input (Unity, undated).



```
public class GenreGeneratorWindow : EditorWindow
{
    FilmController filmController;
    string inputGenre = "";
    public TextAsset filmFile;

    // Add menu named "My Window" to the Window menu
    [MenuItem("Tools/Genre Generation")]
    static void Init()
    {
        // Get existing open window or if none, make a new one:
        GenreGeneratorWindow window = (GenreGeneratorWindow)EditorWindow.GetWindow(typeof(GenreGeneratorWindow), false, "Genre Generator");
        window.Show();
    }

    void OnGUI()
    {
        GUILayout.Label("Output", EditorStyles.boldLabel);
        inputGenre = EditorGUILayout.TextField("Genre", inputGenre);

        if (GUILayout.Button("Start Genre Generation"))
        {
            string[] filme = File.ReadAllLines(Path.Combine(Application.streamingAssetsPath, "Films.txt"));
            List<Film> myFilms = Film.FilmObjectsFromDatafile(filme);
            filmController = new FilmController(myFilms);

            filmController.GetOutputShotsFromGenre(inputGenre);
            filmController.GetMostCommonShots();
            filmController.MatchShotLibraryShotToFilmShot();
            filmController.SerializeFilm();
            AssetDatabase.Refresh();
            EditorUtility.DisplayDialog("Genre Generator", $"Film Sequence of {inputGenre} has been generated.", "Ok");
        }
    }
}
```

Figure 22: A figure showing the updated View class

As can be seen in Figure 22, the View class has a similar functionality to that seen in Version 4. However, the input is now taken from a new item in the menu labelled Tools/Genre Generation, which has a singular text field where a user can enter a genre shown in Figure 23. That string input is then used by the Controller code as demonstrated in Figure 18.



Genre Generator

Film Sequence of Thriller has been generated.

Figure 23: A figure showing the new Genre Generator functionality in Unity

As can be seen in Figure 23, a humanoid asset (Kevin Iglesias, 2020) is now used in place of the cube, with the camera also being positioned to be in the exact same framing as the shot displayed on the canvas in this case a medium shot. This was done to provide further clarification to the user as to how a specific shot would appear in the visual environment. However, the user can still use Unity's camera system to search for other types of shots that they may wish to incorporate. Whilst the shot statistics are no longer there to give system transparency, the use of a functional camera that allows users to experiment instead allows for the creative expression that Resnick believes so important in a system. A final version of this project is therefore shown in Figure 24.

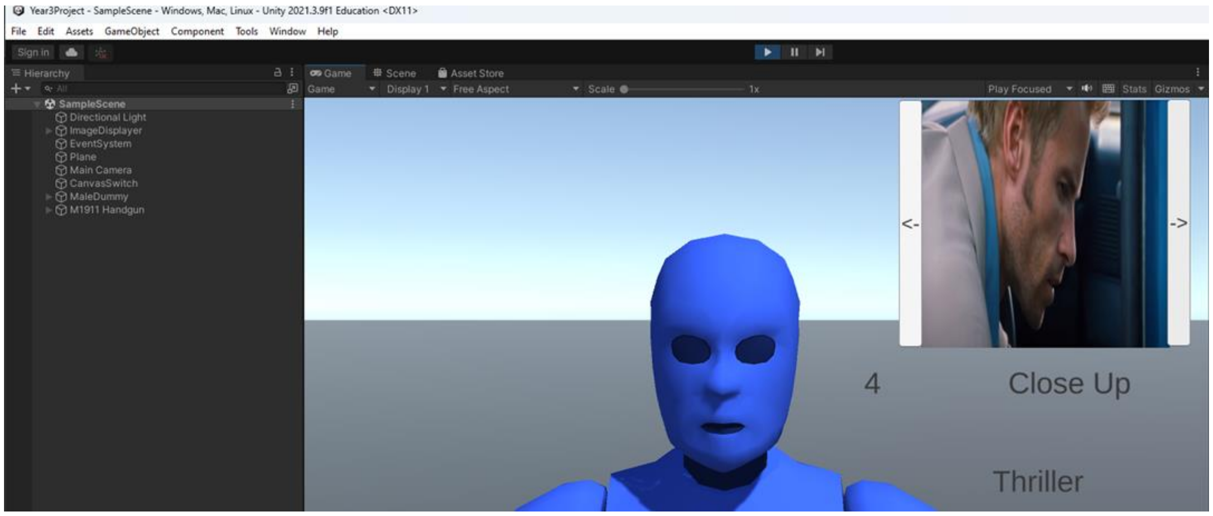


Figure 24: A figure showing the final working version of the project prior to testing

It is believed that all aspects of the requirement analysis have been fulfilled. The user has an output that is clearly understandable as visual clues have been given to help both the novice and the expert alike understand what each film shot represents. Users are also able to experiment with the output via Unity’s camera system, which can be utilised to encourage experimentation for different types of shots. Unity’s UI has been extended to allow user input. A new tab in the editor window has been created to allow for genre generation by the MVC. Finally, a database like system has been designed as the MVC loads in the txt file with information that is handled like a database.

Chapter 6

Results & Discussion

6.1 Black Box Testing

As stated in Chapter 4, the plan to test this project's aim: "Is it possible to create an algorithm that can accurately predict a set of shots a director should employ before creating a film?" was to incorporate user testing, by getting them to use the system to see whether or not they felt satisfied with the shot list that the algorithm generated. Also, as it is a recommendation system, it must also have functionality that allows users the freedom to experiment with the generated output. Finally, as the project is also incorporating a UI, user experience principles must be adhered to. These would also be tested by users. However, before all of this could be done, black box testing was implemented to make sure that the system's functional requirements from the requirement analysis were met, as shown in Table 4.

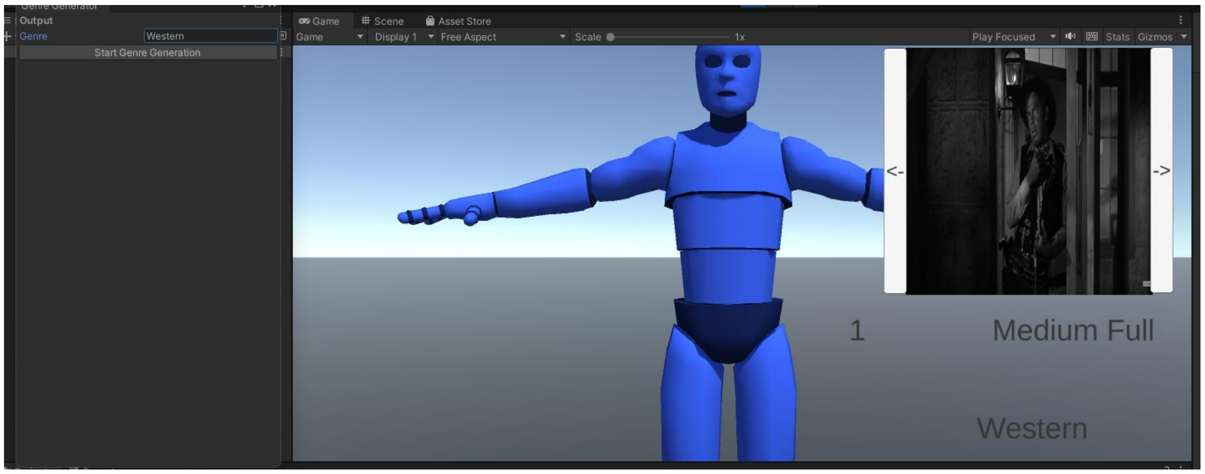
Table 4: A table of the black box tests for this project

Test	**Input Type**	**Expected Output**	**Actual Output**	**Pass/Fail**
The user inputs the genre "Action"	Expected	A shot list is generated of type "Action"	The correct shot list is generated	Pass
The user inputs the genre "Action" in lower case	Borderline	A shot list is generated of type "Action"	Unity throws an error stating "Sequence	Fail

			contains no elements”	
The user inputs the number 2	Erroneous	Unity will throw an error to the user	Unity throws an error stating “Sequence contains no elements”	Pass
The user inputs the genre “Horror”	Erroneous	Unity will throw an error to the user	Unity throws an error stating “Sequence contains no elements”	Pass
The user inputs the genre “Comedy”	Expected	A shot list is generated of type “Comedy”	The correct shot list is generated	Pass
The user inputs the genre “Thriller”	Expected	A shot list is generated of type “Thriller”	The correct shot list is generated	Pass
The User inputs the genre “Western”	Expected	A shot list is generated of type “Western”	The correct shot list is generated	Pass
The user can move the camera using the keys WASD	Expected	When pressing each key, the camera will move left,	The camera moves in the directions expected	Pass

		right, up and down		
The user can zoom the camera in and out with the scroll wheel	Expected	When moving the scroll wheel up and down, the camera can zoom in and out as a result	The camera can zoom in and out as expected	Pass
The user can toggle the interface on/off with F	Expected	The canvas containing the image and text can be turned on and off with F	The canvas toggles on/off as expected	Pass

As can be seen, most of the tests pass. The requirement "Use Unity's UI to allow user input", informed by the objective to decide on a relevant UI for user interaction was successfully implemented. The user can input a specific genre into Unity and a shot list of that genre is outputted to the system. An example of this test for the input "Western" is displayed in Figure 25.

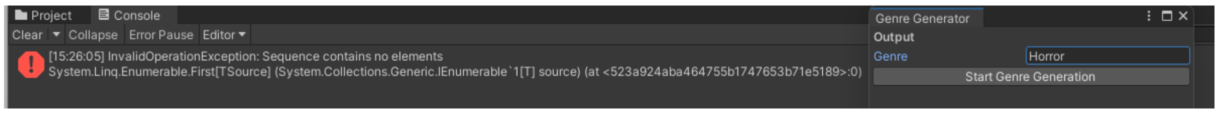

Output

Genre: Western

Figure 25: A figure showing the black box test for generating a "Western"

As can be seen, the correct shot list for the Western has been generated. The requirement "Create an output that is easily understandable to any user" has also been fulfilled. This has been achieved by having two visual references of a shot, in this case a Medium Full. One reference is the use of the shot in a pre-existing film, whilst the other reference is Unity's camera positioned in such a way that it creates a Medium Full shot in Unity's virtual environment. This, therefore, also fulfils the objectives to both implement a system that takes user input to output a finalised shot list as well as displaying relevant images to illustrate to the user what these shots will look like.

Erroneous tests were also given for user inputs, with the expectation that Unity would throw an error i.e. for genres such as "Horror". The input data given to the algorithm would contain no "Horror" films and so no shot list could therefore be generated. An example of this test is shown in Figure 26.


```
[15:26:05] InvalidOperationException: Sequence contains no elements
System.Linq.Enumerable.First[TSource] (System.Collections.Generic.IEnumerable`1[T] source) (at <523a924aba464755b1747653b71e5189>:0)
```

Genre Generator

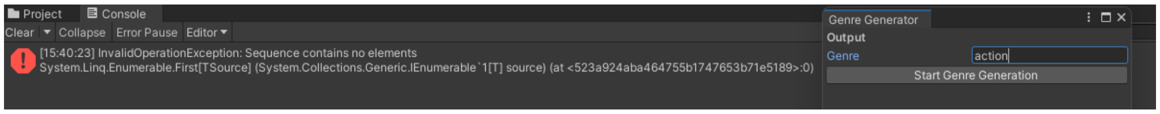
Output

Genre: Horror

Figure 26: A figure to show the result of using "Horror" as an input

As expected, when “Horror” was entered, an error was thrown by Unity, caused by an operation used by LINQ (incorporated in Figure 7). This error was due to the list used by LINQ containing no elements, which would make sense as there are no “Horror” films in the txt file and so, therefore, nothing could be passed through to the list variable “currentShots”. However, this could cause problems when used by a layperson. Nielson states how when creating a good user experience that there should not be any “internal jargon”. A developer can easily understand why the input “Horror” causes Unity to throw an error, however for those unfamiliar with the internal workings of the program, the error thrown may not be understood and may cause confusion. It may, therefore, be more advisable to offer an explanation to the user as to why a genre such as “Horror” cannot be used rather than the error message that is currently displayed.

One black box test resulted in failure. This was when the user input the genre “Action” in lower case. As the input was still relating to generating an action sequence, the expected output would therefore be that the action sequence is displayed. However the system instead treats this input the same way as an invalid genre such as “Horror”, as shown in Figure 27.



```
[15:40:23] InvalidOperationException: Sequence contains no elements
System.Linq.Enumerable.First[TSource] (System.Collections.Generic.IEnumerable`1[T] source) (at <523a924aba464755b1747653b71e5189>:0)
```

Genre: action

Figure 27: A figure to show the result of using “Action” as an input

This means, therefore, that the system is relying on the user to input a string in a specific format, that being that the first letter of the genre is capitalised, whilst all other letters are in lower case. The user may not know that the system expects the input in a specific format. Before testing with users the project should be changed to allow for the inputted genre to be submitted in any case.

As previously stated, this system has been created as a recommendation system. In order to fulfil the stated requirement “Allows users to experiment with the generated output” users are given camera controls, such as using WASD to move the camera and using the scroll wheel to zoom in and out. This could mean moving the camera to create a close up, rather than the medium full recommended in Figure 25. Figure 28 shows the result of the black box test for camera control.

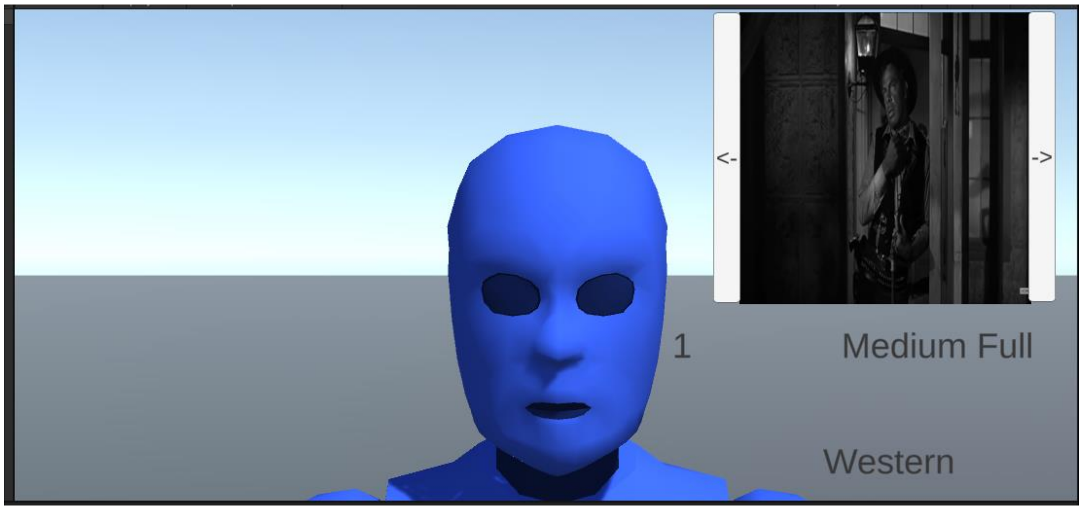


Figure 28: A figure showing how the camera has now moved from the default medium full shot to a close up

As can be seen from Figure 25, the camera defaults to a medium full shot for the first shot of the sequence. However, in Figure 28 the shot is now a close up. This is because the camera has been moved from the medium full position to a close up by the user, meaning that the requirement “Allows users to experiment with the generated output” has been fulfilled.

The final requirement relates to how the system must “Design a database like system to create the outputted shot list”. It is evident that for the genres used as input data; Action, Thriller, Western and Comedy, a shot list is outputted, with the shots displayed being whichever shots are identified as the most common in this database like system. Therefore, this final requirement is also shown to be fully operational as are the first two objectives i.e. to both learn the type of shots that exist in films and to analyse the shot types of ten films to compile as data for the algorithm.

As well as the changes already outlined, some further usability concerns were identified during black box testing, based on Nielson's heuristics. Firstly, whilst the user was able to use camera controls, the functionality for this was not immediately obvious as there was no documentation in the system describing the controls to the user. Also there was no explanation to the user on how to generate shots themselves. Nielson states that while "it's best if the system does not need additional explanation" (1994) it may be necessary "to provide documentation to help users understand how to complete their tasks" (1994). This could be in the form of a separate txt file, that explains the controls available to the user and the way they work within the system. This type of documentation has already been used by the gun asset downloaded for this project (Nokobot, 2020), it has already proven to be a valid way of presenting information to the user. It was also decided that more camera functionality should be included, i.e. having the camera rotate around the scene to allow for more creative expression. Further amendments included adding a text box to describe shots to further increase user understanding and allowing user adjustments to be saved as a pdf.

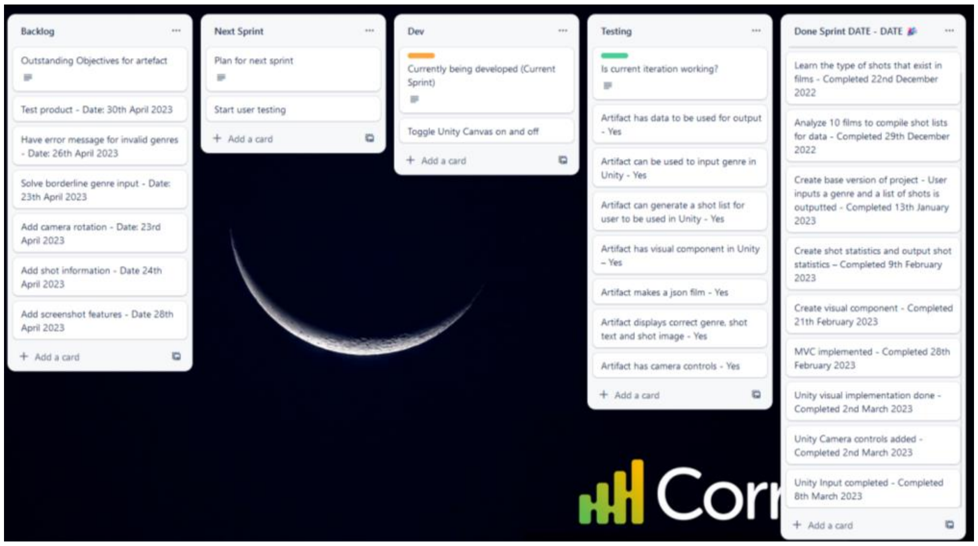
The final consideration before user testing was to increase the dataset to twenty films from the original ten as the current dataset was not truly fulfilling the aim i.e. for some genres, such as "Thriller" and "Comedy" only two films were being considered in the dataset. This would mean that the most common shot could only be either 100% (if the two films contain the same shot) or 50/50 (if the two films contained different shots). This would mean, therefore, that in some cases the algorithm was more likely making a random choice, whereas the stated aim of the algorithm was to be making a choice based on shots that commonly appear. The larger model proposed would mean therefore that each genre has an equal share of five films to analyse, which should minimise the number of times the algorithm makes a random choice. Table 5 lists the further ten films that were used to expand the database.

Table 5: A table to show the ten films added to the project database

Film	**Genre**	**No of Shots**	**Citation**
Hard Boiled	Action	27	(Woo, 1992)
The Untouchables	Thriller	20	(De Palma, 1987)
Hot Shots! Part Deux	Comedy	41	(Abrahams, 1993)
The Outlaw Josey Wales	Western	34	(Eastwood, 1976)
Unforgiven	Western	32	(Eastwood, 1992)
21 Jump Street	Comedy	22	(Lord, Miller, 2012)
The Third Man	Thriller	50	(Reed, 1949)
No Country for Old Men	Thriller	32	(Coen, Coen, 2007)
Léon: The Professional	Action	27	(Besson, 1994)
Kelly’s Heroes	Comedy	34	(Hutton, 1970)

6.2 Final Implementation

A final version was created to address the issues highlighted during black box testing. The changes needed were added to a new Trello board, shown in Figure 29, with Table 6 showcasing the results of black box testing with these new features. Appendix C shows the final version of the project before user testing.



Backlog
- Outstanding Objectives for artefact
- Test product - Date: 30th April 2023
- Have error message for invalid genres - Date: 26th April 2023
- Solve borderline genre input - Date: 23th April 2023
- Add camera rotation - Date: 23rd April 2023
- Add shot information - Date 24th April 2023
- Add screenshot features - Date 28th April 2023

Next Sprint
- Plan for next sprint
- Start user testing

Dev
- Currently being developed (Current Sprint)
- Toggle Unity Canvas on and off

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre in Unity - Yes
- Artifact can generate a shot list for user to be used in Unity - Yes
- Artifact has visual component in Unity – Yes
- Artifact makes a json film - Yes
- Artifact displays correct genre, shot text and shot image - Yes
- Artifact has camera controls - Yes

Done Sprint DATE - DATE
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shot statistics – Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- MVC implemented - Completed 28th February 2023
- Unity visual implementation done - Completed 2nd March 2023
- Unity Camera controls added - Completed 2nd March 2023
- Unity Input completed - Completed 8th March 2023

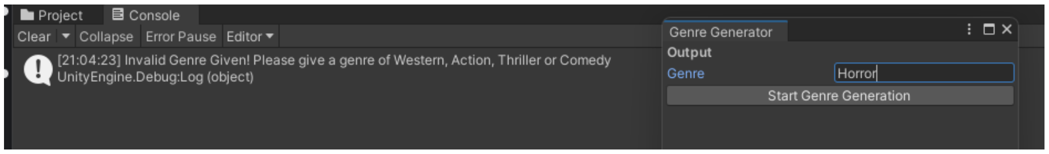
Figure 29: A figure to show the Trello board for the final implementation

Table 6: A table to show black box testing for features in the final implementation

Test	**Input Type**	**Expected Output**	**Actual Output**	**Pass/Fail**
The user inputs the number 2	Erroneous	Unity will throw a custom error to the user	Unity throws an error stating “Invalid Genre Given! Please give a genre of Western, Action, Thriller or Comedy”	Pass

			(shown in Figure 30)	
The user inputs the genre “Horror”	Erroneous	Unity will throw a custom error to the user	Unity throws an error stating “Invalid Genre Given! Please give a genre of Western, Action, Thriller or Comedy”	Pass
The user inputs the genre “Action” in lower case	Borderline	A shot list is generated of type “Action”	The correct shot list is generated	Pass
The user can rotate the camera by holding the left click and dragging the mouse	Expected	The camera rotates in the direction the mouse moves	The camera does rotate in the direction of mouse movement, done via (Devsplorer, 2020)	Pass
The user can get further information on a specific type of shot by pressing I	Expected	When pressing I, information about the current shot is displayed to the user and turned off with I	The information is correct for each current shot and can be turned on and off with I	Pass

The user can create screenshots of current work by pressing C	Expected	When pressing C, a screenshot of the user's current previs can be created and viewed	Screenshots are created of the current previs. Done via (C# Examples, undated)	Pass

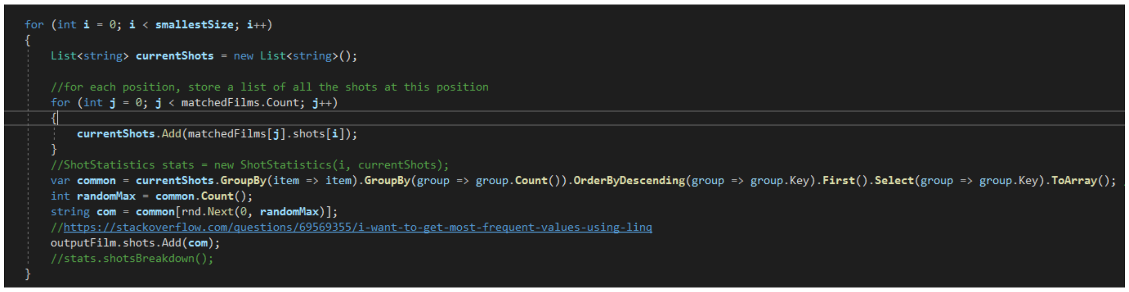


```
[21:04:23] Invalid Genre Given! Please give a genre of Western, Action, Thriller or Comedy
UnityEngine.Debug:Log (object)
```

Genre: Horror

Figure 30: A figure to show improved error messages for invalid genres

All changes outlined are now completed to generate a smoother user experience for testers. It should also be noted that, during this implementation, the way shots were selected in the event of a tie had now changed. Selection would now be a random choice between the shots causing the tie rather than the first shot found by the system, outlined in Figure 31 following a tutorial (StackOverflow, 2021).



```
for (int i = 0; i < smallestSize; i++)
{
    List<string> currentShots = new List<string>();

    //for each position, store a list of all the shots at this position
    for (int j = 0; j < matchedFilms.Count; j++)
    {
        currentShots.Add(matchedFilms[j].shots[i]);
    }
    //ShotStatistics stats = new ShotStatistics(i, currentShots);
    var common = currentShots.GroupBy(item => item).GroupBy(group => group.Count()).OrderByDescending(group => group.Key).First().Select(group => group.Key).ToArray();
    int randomMax = common.Count();
    string com = common[rnd.Next(0, randomMax)];
    //https://stackoverflow.com/questions/69569355/i-want-to-get-most-frequent-values-using-linq
    outputFilm.shots.Add(com);
    //stats.shotsBreakdown();
}
```

Figure 31: A figure to show how tie breaks are now solved

This was done to emulate the goal of system transparency. In an earlier version, the algorithm was explicit in stating how frequently each shot appeared for each part of the sequence, thus offering users more information on the different kinds of shots that could be used, as well as outlining why each shot had been chosen by the algorithm. This method also employs this idea. When a tester continually runs the algorithm for the same genre they may notice that at certain points in the sequence a different shot is generated, whilst other shots remain the same. They can, therefore, infer that the shots that continually change are shots that are not as common, leading users to try and consider using different combinations of shots for this part of the sequence.

With all black box testing complete, the system was deemed adequate to start testing with human participants.

6.3 Results of Human Testing

The final objective was to test with human participants. When testing this project, each participant was asked to complete the steps outlined in Chapter 4.5. Once completed, all questionnaire forms were collected and the data transcribed into the bar charts that are detailed in Appendix D. Responses on the likert scale reflected users' opinions being one of five options; strongly disagree, disagree, neutral, agree, strongly agree. In total, ten participants were asked to test this project, with Appendix D showing an even split between those confident with film shot knowledge and those not confident, meaning the software could be tested on both laypeople and professionals alike. From there, questions could be split into two types:

- Does the software fulfil the aim?
- Does the software deliver a good user experience?

6.3.1 Does the software fulfil the aim?

The aim would be tested by comparing what a user believes their shot list should look like to what the algorithm believes it should look like. Therefore, if the user prefers the algorithm's predictions, then the algorithm succeeds in being able to be used to employ shots in a previs before film creation. If the user instead feels that their own output is better than the algorithm, then the algorithm is not effective in fulfilling this aim. This is shown in Appendix D. In general, all but one user was confident in creating their own shot list, with one opting not to do so due to their lack of experience with film. However, for those that did, all unanimously agreed that the algorithm had a better output than their own creation, meaning that the primary purpose had been fulfilled. As a recommendation system, the algorithm was recommending shots that users found helpful, with one commenting that it "works well at recommending shots I would not have expected or thought of."

6.3.2 Does the software deliver a good user experience?

With the aim having proven to be successful, the other consideration for this software was does it succeed at providing a good user experience i.e. being easily understood by both laypeople and professionals alike. Appendix D asked how well users were able to use the system to differentiate between shots, how well they felt the system could be understood and how well they felt shots were visualised in the system, with the aim being to break down the jargon of film terminology to be more understandable to the layperson. The first and second questions were answered extremely positively, showing that there was no confusion between how different shots looked and that it was easy to understand. However the latter, while still mostly positive, had one tester disagree with the statement that they could visualise how shots would look in the system, while a further two were neutral. This means that while most testers could use the visual guidance provided by Unity to understand how each shot would look in a previs, there is still further work to be done on providing more visual clarity for users who are new to film terminology, even if they understand the general system.

All other questions in Appendix D refer to general user experience, such as ease of use which was considered positive by all, whether it was quick to learn how the system worked, which was again viewed positively, if users felt confident, which most agreed with bar one who was neutral and finally, if the system was cumbersome which most disagreed with, however again one was neutral.

However there were more split opinions when testers were asked if they felt they needed to learn a lot of things before using the system and if the system was for general use, i.e. to create previses for films. Two people agreed that they had needed to learn a lot prior to testing, with one strongly agreeing. This was mainly due to some users not having used the Unity interface prior to testing and thus certain terminology such as “play mode” in the read me document was unfamiliar to them. Since Unity is an interface primarily used for game development, it should be taken into consideration that those from a non-game development background will require a read me document that is not full of “internal jargon” (Nielson, 1994). Eight either agreed or strongly agreed with the question related to general use but two other users were either neutral or disagreed. This could be because the question itself was not fully clear: general use can be considered ambiguous in this context when considering the software used itself is designed for a specialised field, that being film development.

6.3.3 Further Comments

As mentioned in Chapter 4.5, open questions were included in this questionnaire. As expected, not all testers answered this question (40% answered). Those that did offered suggestions for improvements to this project, which will be considered for future development. These included adding more film genres and a dropdown field rather than a text field for genre generation. Along with the positive comment identified in 6.2.1, another user liked the addition of an information button, reinforcing the idea that this system is achieving its function as a recommendation system, especially for the layperson.

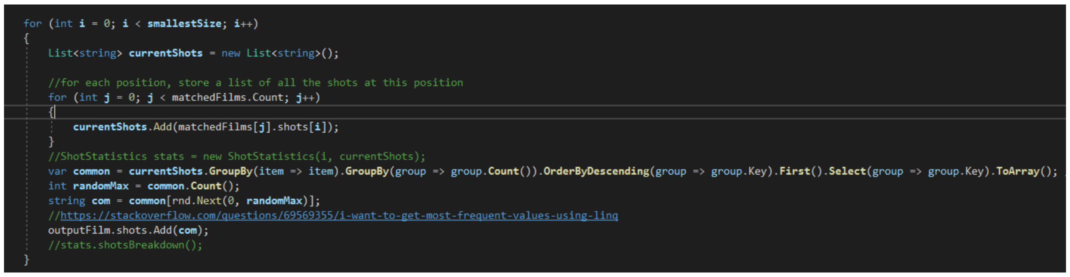
Chapter 7

Conclusion

Overall, the aims and objectives set out to be achieved by the project were all completed to a good standard together with all requirements, as outlined in Chapter 6. Additional requirements identified to create an appropriate recommendation system and a UI with a good user experience were also followed and successfully achieved during development. The aims and objectives were appropriate to the issues identified in the introduction: as the software was designed to aid future directors in choosing suitable film shots. An algorithmic solution was deemed necessary, with the purpose of this algorithm being to identify the most common film shots for a director to choose. All objectives were therefore created bearing this algorithm in mind; to create the dataset used by the algorithm the specific film shot used must first be learned and films needed to also be analysed to create data for the algorithm. To facilitate the user experience requirements, images were needed to provide visual indications to the user as to how a shot would look. A good UI was required to facilitate UXD heuristics. To assess how effective the software was in achieving the aim, the artefact needed to be tested by users to generate data that could support or hinder the question outlined in the aim, also testing whether or not the software was effective as a recommendation system and if it successfully provided a good user experience. The results gathered in Chapter 6 show that this was predominately achieved.

The initial methodology proposed for this project was not followed, due to the initial aims and objectives changing over time and the methodology therefore needing to adapt. As can be seen in Appendix A, the use of Scrumban has been successfully implemented over the course of the project to keep track of current objectives (shown in product backlog) whilst also using the fluidity of agile to continually adapt the product backlog when potential improvements or requirements were discovered during implementation.

This project has used a database like system to generate an algorithm used to predict a set of film shots with the purpose of creating a previs that can be used by aspiring directors. Whilst the use of this type of algorithm has been successful, there are improvements to consider. Firstly, as described in Chapter 3, the use of a database like system was ideally imagined to be of an O(N) complexity, however in practice, as can be seen in Figure 32, the complexity of this project is instead $O(N^2)$ due to a nested for loop produced when generating the shot list for the user to use.



```
for (int i = 0; i < smallestSize; i++)
{
    List<string> currentShots = new List<string>();

    //for each position, store a list of all the shots at this position
    for (int j = 0; j < matchedFilms.Count; j++)
    {
        currentShots.Add(matchedFilms[j].shots[i]);
    }
    //ShotStatistics stats = new ShotStatistics(i, currentShots);
    var common = currentShots.GroupBy(item => item).GroupBy(group => group.Count()).OrderByDescending(group => group.Key).First().Select(group => group.Key).ToArray();
    int randomMax = common.Count();
    string com = common[rnd.Next(0, randomMax)];
    //https://stackoverflow.com/questions/69569355/i-want-to-get-most-frequent-values-using-linq
    outputFilm.shots.Add(com);
    //stats.shotsBreakdown();
}
```

Figure 32: A figure to show the nested for loop that exists in this project

It was argued in Chapter 3 that, due to the relatively small dataset being used, a complexity of O(N) was considered to not adversely affect the project. This is still considered to be the case even with a quadratic complexity. However, it is clear that for a much larger dataset, such as a 100 film dataset, that the time taken to generate the film list would start to become untenable. Furthermore, the data file must follow a specific format in order for the data to be properly handled by the algorithm. This would become time consuming as the number of films added to this file increases. Although SQL was argued to not be needed for the scope of this project, on reflection it may have made the organisation of data more efficient than the method currently used and may be able to reach the O(N) complexity desired when creating queries, so should be considered if implementing this project with a larger dataset.

During the literature review, more research may have been advisable to investigate different ways to generate the shot list for the user. Whilst it was argued that machine learning may not have been a fully advisable solution due to the potential of having too small an accuracy, the machine learning paradigm proposed, a CNN has a complexity outlined in Figure 33.

$$O\left(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right)$$

Figure 33: A figure showing the time complexity of a CNN (He et al., 2015, 2)

As s and m are constants in this equation, the complexity simplifies down to O(N), the same as was proposed in the requirement analysis and, therefore, more efficient than the current implementation. This means that further research to try and increase the accuracy of machine learning should be carried out to correctly identify film shots for the input data. This would then be used instead as the input data to generate film lists. Whilst great care was taken to ensure that no film shots had been misclassified during this research, it should be acknowledged that there is still the potential that human error may have found its way into the system, so by having more accurate machine learning models the risk of this can also be significantly reduced. Another implementation of this project could try to implement a machine learning model to aim for better accuracy than discovered by Vacchetti et al., improving the time complexity for larger datasets and removing human error. However, if this route was taken, research would be needed on finding a dataset for training, decide which machine learning algorithm should be utilised and finally whether or not inaccuracy can ever truly be eradicated by machine learning.

Further research should be considered to examine the context in which the different shots are used and to incorporate this when creating the shot list. Bowen, notes how shots such as a close ups to “bring the subject inside a viewer’s personal space – in a good way if the viewer likes the subject and in a bad way if he or she does not” (2017, 14). In addition Heiderich explains when certain shots should be used “Moving from long (wide) to close shots is a trade-off between showing informative visuals or intimate emotions. You can’t have more of one without giving up an equal amount of the other” (2012, 6). This could mean, for instance, that a director may wish to show a close up for character emotions and a wide shot for establishing potential locations. Many directors have a distinct visual style that often influences the types of shots they use. Wes Anderson for example has “perfected a type of shot…. a static, flat-looking, medium-long (medium full) or long (wide) shot” (MacDowell, 2012, 4). The current model proposed is purely statistical and only focuses on how common each shot is at each position, with the subtext being that these common shots are the best shots to use due to their frequency. Further research should also be conducted, therefore, to examine what shot would best lead on to another based on either a director’s given style or the current context of the film analysed.

One final consideration to develop the system still further is to also include more of a filmmaker’s lexicon into the system. Currently, only a specific subset of shots, such as the close up, have been included as input data for the system. However, there are other aspects of film to consider, for instance the angle at which a shot is filmed, i.e. a low angled shot. This would help to further the layperson’s understanding of filmmaking.

The stated aim of this system was to create an algorithm that could recommend film shots to a user, with the purpose of making the act of film making more accessible to the layperson. The results gathered show that users found that the software recommended a better shot list than the one they had initially created, with the different shots also being easy to differentiate between. However, in creating a model that focuses on the most commonly used shots, is the creative freedom of the layperson being restricted if they all choose to use the same generated shot list which they perceive as being better than their own? Would this, therefore, risk films becoming more homogeneous and stifle individual expression? It may, therefore, be beneficial to conduct further research into whether or not this project facilitates the creative freedom it hoped for or instead stifles it given the tools supplied to encourage this.

Word Count

12144

References

Abrahams, J. (dir.) (1993) *Hot Shots! Part Deux* [film]. 20th Century Fox.

Anderson, W. (dir.) (2014) *The Grand Budapest Hotel* [film]. Fox Searchlight Pictures.

Armitage, G. (dir.) (1997) *Grosse Pointe Blank* [film]. Buena Vista Pictures Distribution.

Arnold, V., Clark, N., Collier, P.A., Leech, S.A. and Sutton, S.G., 2006. The differential use and effect of knowledge-based system explanations in novice and expert judgment decisions. *Mis Quarterly*, pp.79-97.

Atlassian (2020) *Trello* [software]. New York: Atlassian. Available from https://trello.com/?&aceid=&adposition=&adgroup=150132215251&campaign=19250238417&creative=641300559579&device=c&keyword=trello&matchtype=e&network=g&placement=&ds_kids=p74526191747&ds_e=GOOGLE&ds_eid=700000001557344&ds_e1=GOOGLE&gad=1&gclid=EAIaIQobChMIg_qz9dnP_gIVGr_tCh0cwQzcEAAYASAAEgKtXPD_BwE&gclsrc=aw.ds [accessed 25 January 2023].

Besson, L. (dir.) (1994) *Léon: The Professional* [film]. Gaumont Buena Vista International.

Bowen, C.J., 2017. *Grammar of the Shot*. Taylor & Francis.

Breitinger, F., Baier, H. and White, D., 2014. On the database lookup problem of approximate matching. *Digital Investigation, 11*, pp.S1-S9.

Brown, B., 2016. *Cinematography Theory and Practice: Imagemaking for Cinematographers & Directors*. Routledge. [accessed 25th January 2023].

Bucanek, J., 2009. Model-view-controller pattern. *Learn Objective-C for Java Developers*, pp.353-402.

Byrne, J. (2022) 'Developing Software to Create High Quality Game Cinematics' *CMP3753M: Project.* University of Lincoln. Unpublished assignment.

Byrne, J. (2023) 'Literature Review and Progress Update', *CMP3753M: Project.* University of Lincoln. Unpublished assignment.

C# Examples (undated) *Delete All Files (*.*) [C#]*. C# Examples. Available from https://www.csharp-examples.net/delete-all-files/ [accessed 28 April 2023].

Case, D., (2013). Film technology in post production. Taylor & Francis. [accessed 28 October 2022].

Coen, J. Coen, E. (dirs.) (2007) *No Country for Old Men* [film]. Paramount Vantage.

Connolly, T.M. and Begg, C.E., 2005. *Database systems: a practical approach to design, implementation, and management*. Pearson Education.

De Goussencourt, T., Dellac, J. and Bertolino, P., 2015. A game engine as a generic platform for real-time previz-on-set in cinema visual effects. In *Advanced Concepts for Intelligent Vision Systems: 16th International Conference, ACIVS 2015, Catania, Italy, October 26-29, 2015. Proceedings 16* (pp. 883-894). Springer International Publishing.

De Palma, B. (dir.) (1987). *The Untouchables* [film]. Paramount Pictures.

Devsplorer (2020) *How to Rotate Camera With Mouse Drag In X Y Axies In Unity | Unity 3D Tutorial* [video]. Available from https://www.youtube.com/watch?v=FIiKuP-9KuY [accessed 24 April 2023].

Dillon, E., Anderson, M. and Brown, M., 2012, March. Comparing mental models of novice programmers when using visual and command line environments. In *Proceedings of the 50th Annual Southeast Regional Conference* (pp. 142-147).

Eastwood, C. (dir.) (1976) *The Outlaw Joesy Wales* [film]. Warner Bros.

Eastwood, C. (dir.) (1992) *Unforgiven* [film]. Warner Bros.

Endfield, C. (dir.) (1964) *Zulu* [film]. Paramount Pictures.

Epic Games (2022) *Unreal Engine 5* [software]. Cary: Epic Games. Available from https://www.unrealengine.com/en-US/unreal-engine-5 [accessed 4 May 2023].

Ferreira, J., Noble, J. and Biddle, R., 2007, August. Agile development iterations and UI design. In *Agile 2007 (AGILE 2007)* (pp. 50-58). IEEE.

Ford, J. (dir.) (1962) *The Man Who Shot Liberty Valance* [film]. Paramount Pictures.

Game Developer Training (2022) *[Unity] Beginner guide – Moving the camera with WASD* [video]. Available from https://www.youtube.com/watch?v=k_-sOqkvoI8 [accessed 2 March 2023].

Github (undated) *Let's build from here.* San Francisco: Github. Available from https://github.com/ [accessed 10 April 2023].

He, K. and Sun, J., 2015. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 5353-5360).

Heiderich, T., 2012. Cinematography techniques: The different types of shots in film. *Videomakers. Accessed September*, *22*, p.2020.

Heiderich, T., 2012. Cinematography techniques: The different types of shots in film. *Videomakers. Accessed September*, *22*, p.2020.

Hobbs, R., 2019. Transgression as creative freedom and Creative Control in the Media Production Classroom. *International Electronic Journal of Elementary Education*, *11*(3), pp.207-215.

Hutton, B. (dir.) (1970) *Kelly's Heroes* [film]. Metro-Goldwyn-Mayer.

Kevin Iglesias (2020) *3D Character Dummy* [asset]. Unity. Available from https://assetstore.unity.com/packages/3d/characters/humanoids/humans/3d-character-dummy-178395 [accessed 7 March 2023].

Kwan, D. Scheinet, D. (dirs.) (2022) *Everything Everywhere All At Once* [film]. A24.

Leone, S. (dir.) (1966) *The Good, the Bad and the Ugly* [film]. Produzioni Europee Associate.

Lord, P. Miller, C. (dirs.) (2012) *21 Jump Street* [film]. Columbia Pictures. Metro-Goldwyn-Mayer.

MacDowell, J., 2012. Wes Anderson, tone and the quirky sensibility. *New Review of Film and Television Studies*, *10*(1), pp.6-27.

Mahalakshmi, M. and Sundararajan, M., 2013. Traditional SDLC vs scrum methodology–a comparative study. *International Journal of Emerging Technology and Advanced Engineering*, *3*(6), pp.192-196.

McKay, E.N., 2013. *UI is communication: How to design intuitive, user centered interfaces by focusing on effective communication*. Newnes.

Melton, J., 1996. Sql language summary. *Acm Computing Surveys (CSUR)*, *28*(1), pp.141-143.

Microsoft (undated) *Process Class (System.Diagnostics)* Redmond: Microsoft. Available from https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process?view=net-8.0 [accessed 9 April 2023].

Najafi, M. and Toyoshiba, L., 2008, August. Two case studies of user experience design and agile development. In *Agile 2008 Conference* (pp. 531-536). IEEE.

Nielsen, J., 1994 10. Usability heuristics for user interface design.

Nitsche, M., 2008, November. Experiments in the use of game technology for pre-visualization. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share* (pp. 160-165).

Nokobot (2020) *Modern Guns: Handgun* [asset]. Unity. Available from https://assetstore.unity.com/packages/3d/props/guns/modern-guns-handgun-129821 [accessed 8 March 2023].

Nolan, C. (dir.) (2000) *Memento* [film]. Newmarket.

Petricioli, L. and Fertalj, K., 2022, May. Agile Software Development Methods and Hybridization Possibilities Beyond Scrumban. In *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)* (pp. 1093-1098). IEEE.

Pries, K.H. and Quigley, J.M., 2010. *Scrum project management*. CRC press.

Prosser, J.D., 2012. Visual methodology. *Collecting and interpreting qualitative materials*, *177*.

Ravid, S.A., 1999. Information, blockbusters, and stars: A study of the film industry. *The Journal of Business*, *72*(4), pp.463-492.

Reed, C. (dir.) (1949) *The Third Man* [film]. British Lion Film Corporation.

Reja, U., Manfreda, K.L., Hlebec, V. and Vehovar, V., 2003. Open-ended vs. close-ended questions in web questionnaires. *Developments in applied statistics*, *19*(1), pp.159-177.

Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T. and Eisenberg, M., 2005. Design principles for tools to support creative thinking.

Robillard, M., Walker, R. and Zimmermann, T., 2009. Recommendation systems for software engineering. *IEEE software*, *27*(4), pp.80-86.

Ronfard, R., Gandhi, V., Boiron, L. and Murukutla, V.A., 2015. The prose storyboard language: A tool for annotating and directing movies. *arXiv preprint arXiv:1508.07593*. [accessed 24th January 2023].

Schlesinger, J. (dir.) (1976) *Marathon Man* [film]. Paramount Pictures.

Silberschatz, A., Korth, H.F. and Sudarshan, S., 2011. Database system concepts.

Srivastava, A., Bhardwaj, S. and Saraswat, S., 2017, May. SCRUM model for agile methodology. In *2017 International Conference on Computing, Communication and Automation (ICCCA)* (pp. 864-869). IEEE.

StackOverflow (2008) *Find the most occurring number in a List<int>* StackOverflow Available from https://stackoverflow.com/questions/355945/find-the-most-occurring-number-in-a-listint [accessed 10 January 2023].

StackOverflow (2009) *A method to count occurrences in a list* StackOverflow Available from https://stackoverflow.com/questions/1139181/a-method-to-count-occurrences-in-a-list [accessed 9 February 2023].

StackOverflow (2021) *I want to get the most frequent values using LINQ*. StackOverflow Available from https://stackoverflow.com/questions/69569355/i-want-to-get-most-frequent-values-using-linq [accessed 22 April 2023].

Stahelski, C. (dir.) (2014) *John Wick* [film]. Summit Entertainment.

Studiobinder (2020) *Ultimate Guide to Camera Shots: Every Shot Size Explained [The Shot List, Ep 1]* [video]. Available from https://www.youtube.com/watch?v=AyML8xuKfoc&list=PLEzQZpmbzckV0_a2QCO2qF9Yfe-LKSDha [accessed 20 December 2022].

Studiobinder (undated) *Different types of camera shots in film*. Studiobinder. Available from https://www.studiobinder.com/blog/types-of-camera-shots-sizes-in-film [accessed 27 April 2023].

Tarantino, Q. (dir.) (1992) *Reservoir Dogs* [film]. Miramax Films.

Tarantino, Q. (dir.) (2012) *Django Unchained* [film]. Columbia Pictures.

TechCrunch (2023) *Github says it now has 100M active users*. San Francisco: TechCrunch. Available from: https://techcrunch.com/2023/01/26/github-says-it-now-has-100m-active-users/?guccounter=1&guce_referrer=aHR0cHM6Ly9kdWNrZHVja2dvLmNvbS8&guce_referrer_sig=AQAAAFz7DQpD_uon-qsXHSUU4n6j3CbisTMX600BxN83fpH2VpZlKYysZPLUh7oPBMANB7mj3h6v690Sv0Pc_E82aEhM_CGfFnDUTXTtRrYKToeNynf7M22FCywtxva3viaTai5iFC02j_nMdwdFvFJ7oqxGBOmjPASa8zP5wSrF8fQc [accessed 10 April 2023].

The Numbers (2022) *Everything Everywhere All At Once (2022)* Beverly Hills: The Numbers. Available from https://www.the-numbers.com/movie/Everything-Everywhere-All-At-Once-(2022)#tab=summary [accessed 7 March 2023].

Unity (2012) *How do I make the camera zoom in and out with the mouse wheel?* Unity. Available from https://answers.unity.com/questions/218347/how-do-i-make-the-camera-zoom-in-and-out-with-the.html [accessed 2 March 2023].

Unity (2014) *How to read .json file.* Unity. Available from https://forum.unity.com/threads/how-to-read-json-file.401306/ [accessed 2 March 2023].

Unity (2021) *2021 Gaming Report.* Unity. Available from https://create.unity.com/2021-game-report [accessed 6 April 2023].

Unity (undated) *EditorWindow* Unity. Available from https://docs.unity3d.com/ScriptReference/EditorWindow.html [accessed 8 March 2023].

Unity (undated) *Unity – Scripting API: Editor*. Unity. Available from https://docs.unity3d.com/ScriptReference/Editor.html [accessed 6 April 2023].

Unity Technologies (2023) *Unity* [software]. San Francisco: Unity Technologies. Available from https://unity.com/download [accessed 4 May 2023].

Vacchetti, B. and Cerquitelli, T., 2022. Cinematographic Shot Classification with Deep Ensemble Learning. *Electronics*, *11*(10), p.1570.

Van Den Brink, H., Van Der Leek, R. and Visser, J., 2007, September. Quality assessment for embedded SQL. In *Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007)* (pp. 163-170). IEEE.

Woo, J. (dir.) (1992) *Hard Boiled* [film]. Golden Princess Film Production.

Wu, H.Y. and Christie, M., 2015, May. Stylistic patterns for generating cinematographic sequences. In *4th Workshop on Intelligent Cinematography and Editing Co-Located w/Eurographics 2015*.

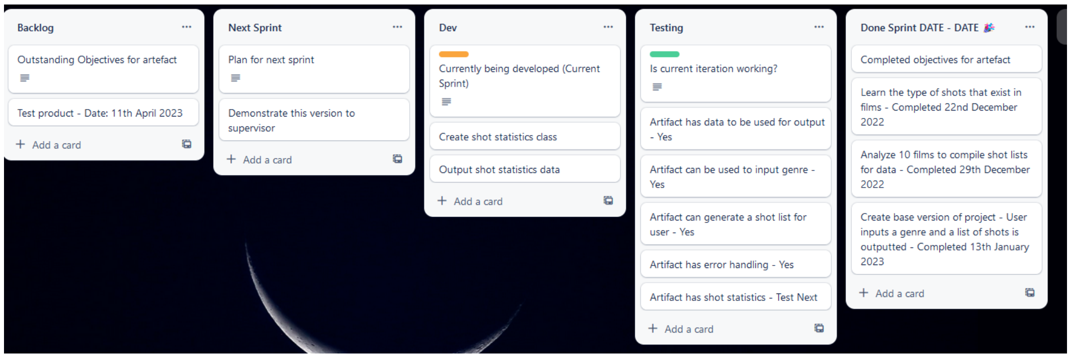
Wu, H.Y., Palù, F., Ranon, R. and Christie, M., 2018. Thinking like a director: Film editing patterns for virtual cinematographic storytelling. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 14*(4), pp.1-22.

Zhao, R., Benbasat, I. and Cavusoglu, H., 2019. DO USERS ALWAYS WANT TO KNOW MORE? INVESTIGATING THE RELATIONSHIP BETWEEN SYSTEM TRANSPARENCY AND USERS'TRUST IN ADVICE-GIVING SYSTEMS.

Appendix A: Trello Boards

This appendix details all Trello boards used during the development of this assignment after the first iteration. Each sub section details the start and end of each iteration.

A.1 Second iteration



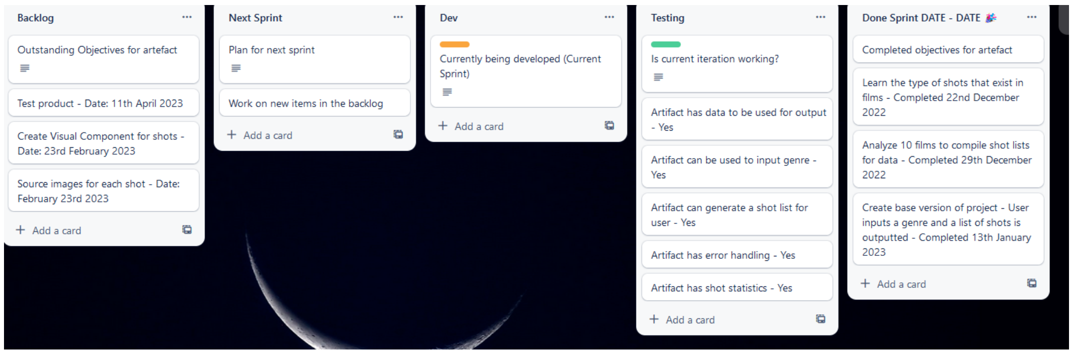
Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- Add a card

Next Sprint
- Plan for next sprint
- Demonstrate this version to supervisor
- Add a card

Dev
- Currently being developed (Current Sprint)
- Create shot statistics class
- Output shot statistics data
- Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user - Yes
- Artifact has error handling - Yes
- Artifact has shot statistics - Test Next
- Add a card

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Add a card



Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- Create Visual Component for shots - Date: 23rd February 2023
- Source images for each shot - Date: February 23rd 2023
- Add a card

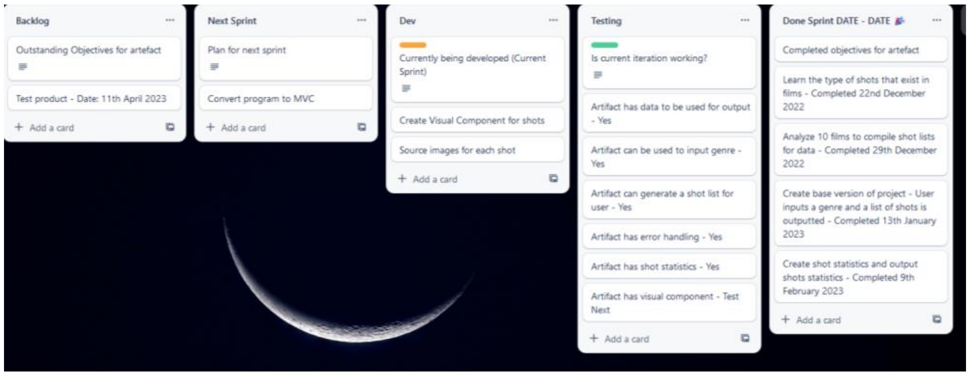
Next Sprint
- Plan for next sprint
- Work on new items in the backlog
- Add a card

Dev
- Currently being developed (Current Sprint)
- Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user - Yes
- Artifact has error handling - Yes
- Artifact has shot statistics - Yes
- Add a card

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Add a card

A.2 Third iteration

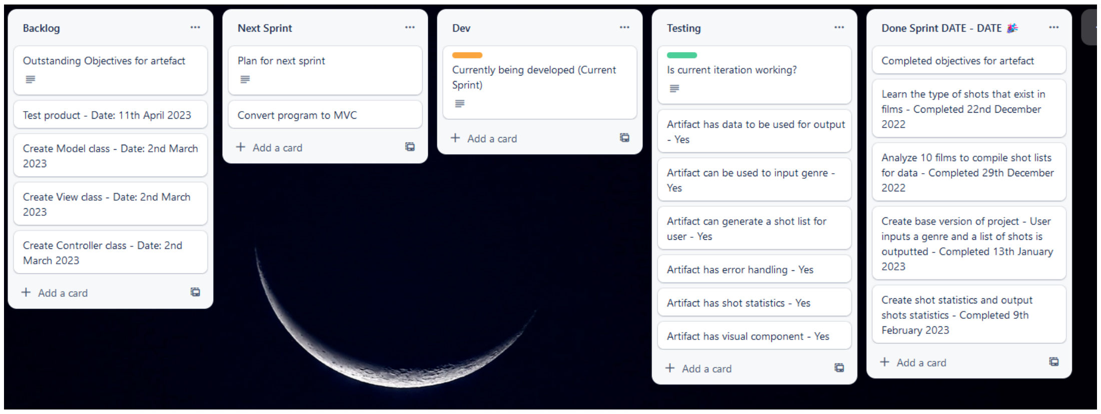

Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- Add a card

Next Sprint
- Plan for next sprint
- Convert program to MVC
- Add a card

Dev
- Currently being developed (Current Sprint)
- Create Visual Component for shots
- Source images for each shot
- Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user - Yes
- Artifact has error handling - Yes
- Artifact has shot statistics - Yes
- Artifact has visual component - Test Next
- Add a card

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shots statistics - Completed 9th February 2023
- Add a card


Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- Create Model class - Date: 2nd March 2023
- Create View class - Date: 2nd March 2023
- Create Controller class - Date: 2nd March 2023
- Add a card

Next Sprint
- Plan for next sprint
- Convert program to MVC
- Add a card

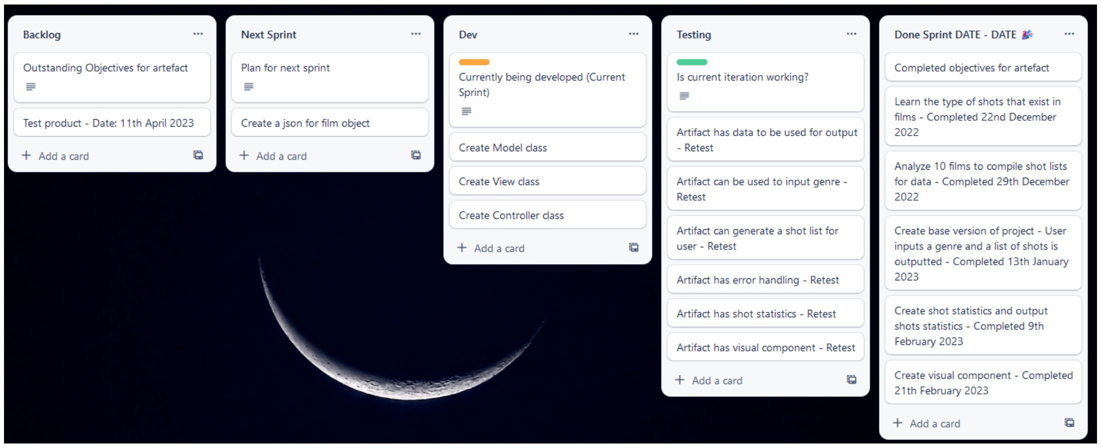
Dev
- Currently being developed (Current Sprint)
- Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user - Yes
- Artifact has error handling - Yes
- Artifact has shot statistics - Yes
- Artifact has visual component - Yes
- Add a card

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shots statistics - Completed 9th February 2023
- Add a card

A.3 Fourth iteration

It was discovered that an extra feature was during the MVC creation, so the fourth iteration had an updated product backlog during development.

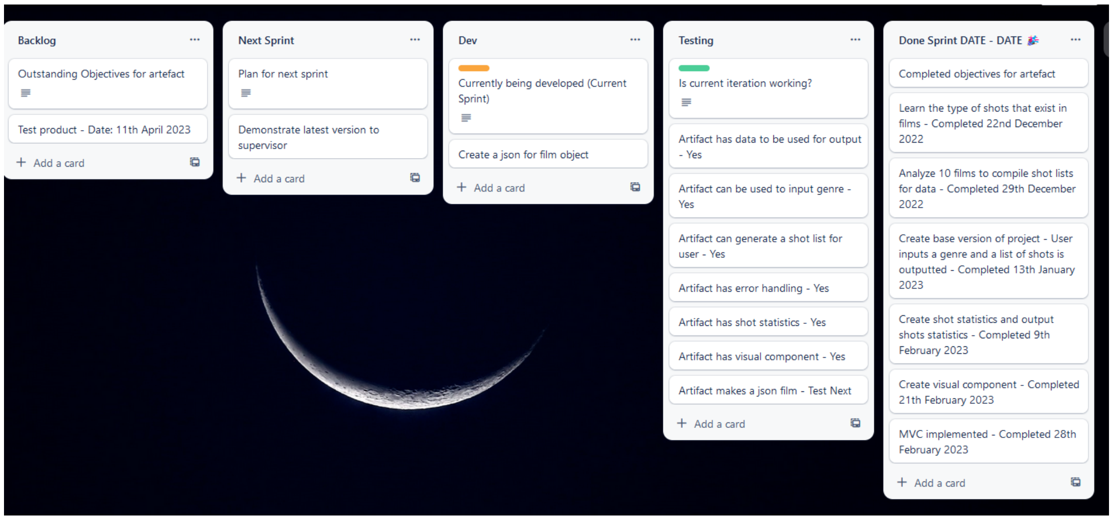

Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- Add a card

Next Sprint
- Plan for next sprint
- Create a json for film object
- Add a card

Dev
- Currently being developed (Current Sprint)
- Create Model class
- Create View class
- Create Controller class
- Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Retest
- Artifact can be used to input genre - Retest
- Artifact can generate a shot list for user - Retest
- Artifact has error handling - Retest
- Artifact has shot statistics - Retest
- Artifact has visual component - Retest
- Add a card

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shots statistics - Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- Add a card



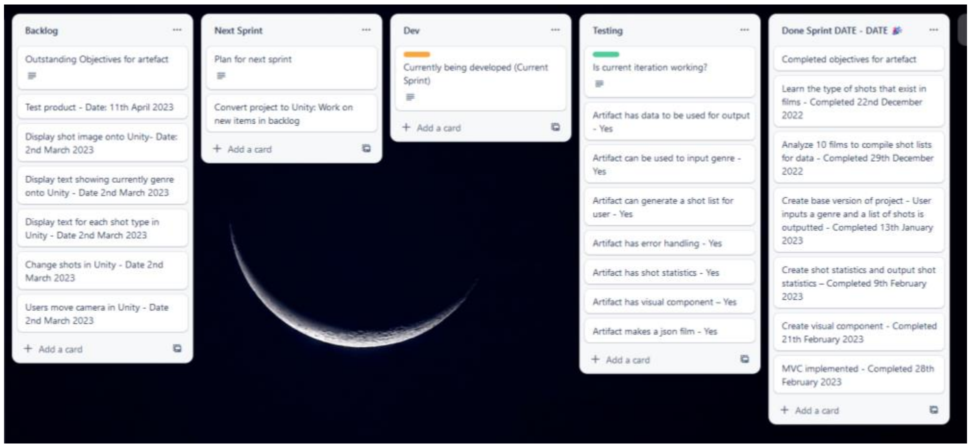
Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023

Next Sprint
- Plan for next sprint
- Demonstrate latest version to supervisor

Dev
- Currently being developed (Current Sprint)
- Create a json for film object

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user - Yes
- Artifact has error handling - Yes
- Artifact has shot statistics - Yes
- Artifact has visual component - Yes
- Artifact makes a json film - Test Next

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shots statistics - Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- MVC implemented - Completed 28th February 2023



Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- Display shot image onto Unity- Date: 2nd March 2023
- Display text showing currently genre onto Unity - Date 2nd March 2023
- Display text for each shot type in Unity - Date 2nd March 2023
- Change shots in Unity - Date 2nd March 2023
- Users move camera in Unity - Date 2nd March 2023

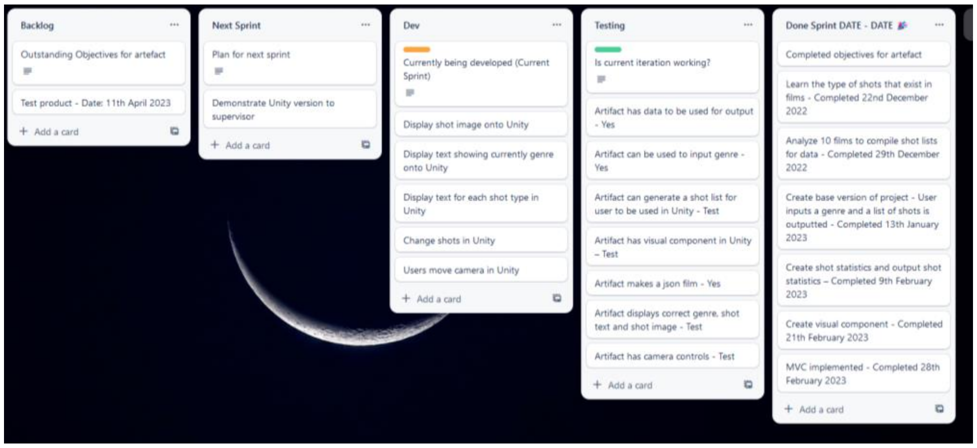
Next Sprint
- Plan for next sprint
- Convert project to Unity: Work on new items in backlog

Dev
- Currently being developed (Current Sprint)

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user - Yes
- Artifact has error handling - Yes
- Artifact has shot statistics - Yes
- Artifact has visual component – Yes
- Artifact makes a json film - Yes

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shot statistics – Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- MVC implemented - Completed 28th February 2023

A.4 Fifth iteration



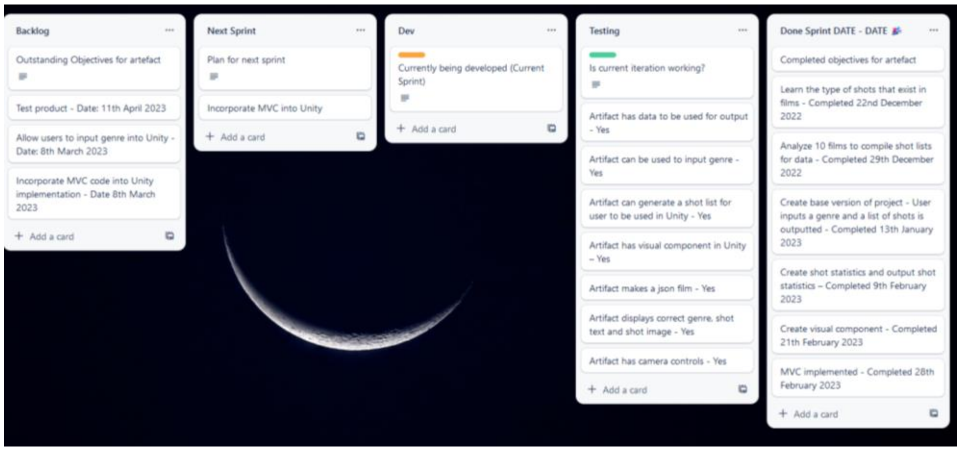
Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023

Next Sprint
- Plan for next sprint
- Demonstrate Unity version to supervisor

Dev
- Currently being developed (Current Sprint)
- Display shot image onto Unity
- Display text showing currently genre onto Unity
- Display text for each shot type in Unity
- Change shots in Unity
- Users move camera in Unity

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user to be used in Unity - Test
- Artifact has visual component in Unity – Test
- Artifact makes a json film - Yes
- Artifact displays correct genre, shot text and shot image - Test
- Artifact has camera controls - Test

Done Sprint DATE - DATE 🎉
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shot statistics – Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- MVC implemented - Completed 28th February 2023



Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- Allow users to input genre into Unity - Date: 8th March 2023
- Incorporate MVC code into Unity implementation - Date 8th March 2023
- + Add a card

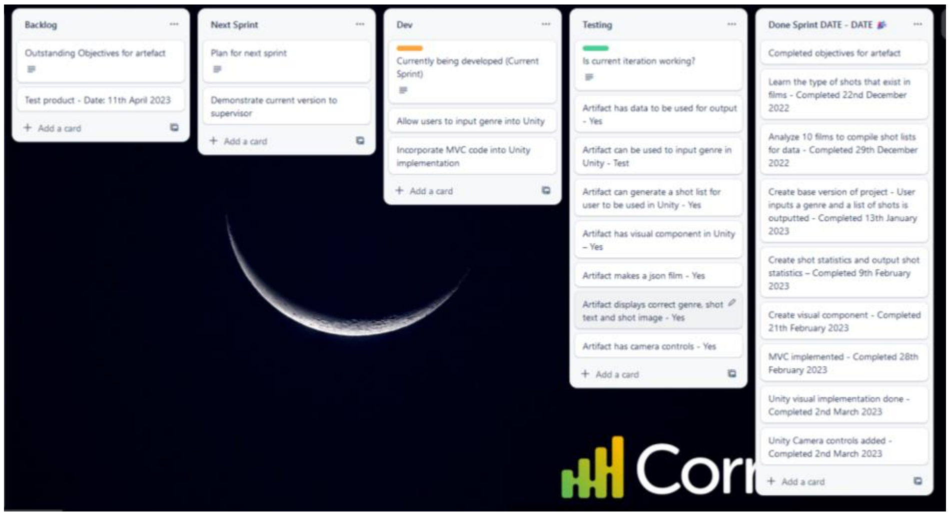
Next Sprint
- Plan for next sprint
- Incorporate MVC into Unity
- + Add a card

Dev
- Currently being developed (Current Sprint)
- + Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre - Yes
- Artifact can generate a shot list for user to be used in Unity - Yes
- Artifact has visual component in Unity – Yes
- Artifact makes a json film - Yes
- Artifact displays correct genre, shot text and shot image - Yes
- Artifact has camera controls - Yes
- + Add a card

Done Sprint DATE - DATE
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shot statistics – Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- MVC implemented - Completed 28th February 2023
- + Add a card

A.5 Sixth iteration



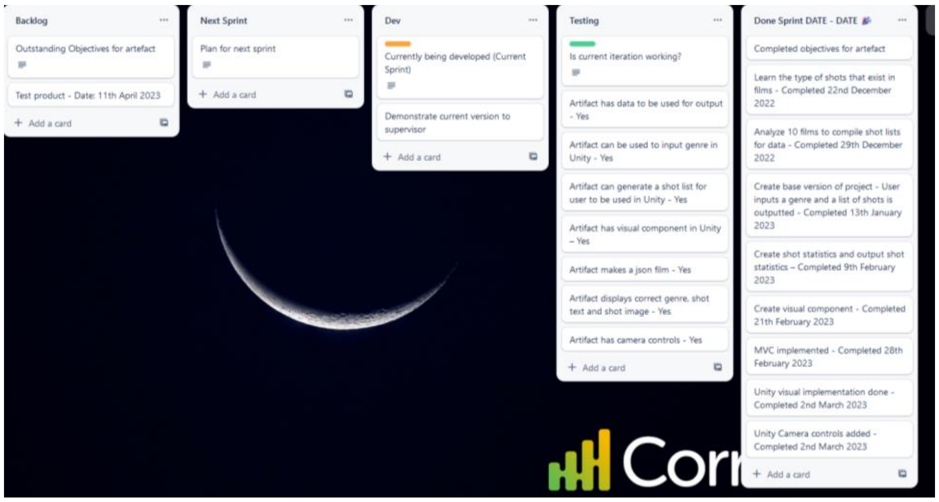
Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- + Add a card

Next Sprint
- Plan for next sprint
- Demonstrate current version to supervisor
- + Add a card

Dev
- Currently being developed (Current Sprint)
- Allow users to input genre into Unity
- Incorporate MVC code into Unity implementation
- + Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre in Unity - Test
- Artifact can generate a shot list for user to be used in Unity - Yes
- Artifact has visual component in Unity – Yes
- Artifact makes a json film - Yes
- Artifact displays correct genre, shot text and shot image - Yes
- Artifact has camera controls - Yes
- + Add a card

Done Sprint DATE - DATE
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shot statistics – Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- MVC implemented - Completed 28th February 2023
- Unity visual implementation done - Completed 2nd March 2023
- Unity Camera controls added - Completed 2nd March 2023
- + Add a card



Backlog
- Outstanding Objectives for artefact
- Test product - Date: 11th April 2023
- + Add a card

Next Sprint
- Plan for next sprint
- + Add a card

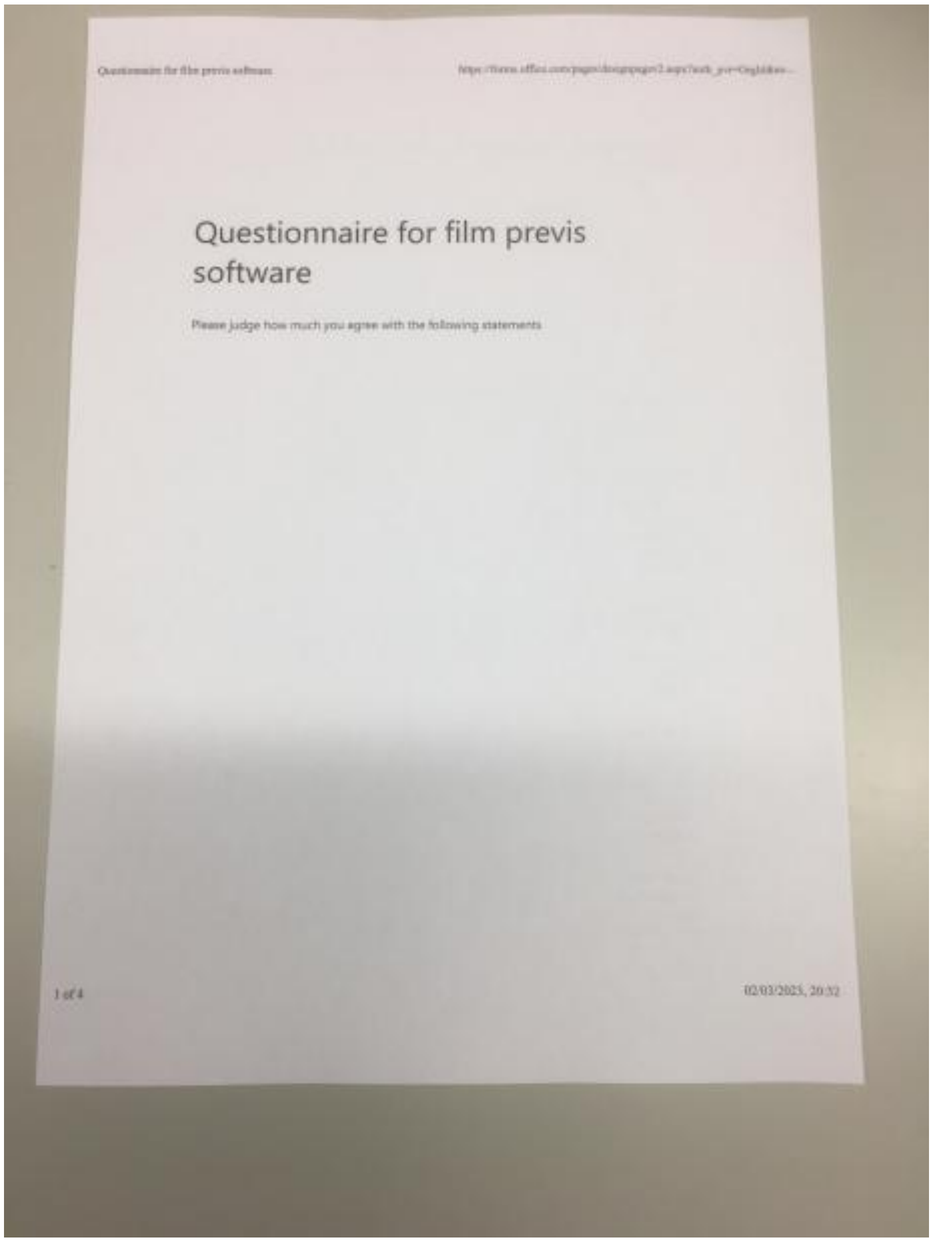
Dev
- Currently being developed (Current Sprint)
- Demonstrate current version to supervisor
- + Add a card

Testing
- Is current iteration working?
- Artifact has data to be used for output - Yes
- Artifact can be used to input genre in Unity - Yes
- Artifact can generate a shot list for user to be used in Unity - Yes
- Artifact has visual component in Unity – Yes
- Artifact makes a json film - Yes
- Artifact displays correct genre, shot text and shot image - Yes
- Artifact has camera controls - Yes
- + Add a card

Done Sprint DATE - DATE
- Completed objectives for artefact
- Learn the type of shots that exist in films - Completed 22nd December 2022
- Analyze 10 films to compile shot lists for data - Completed 29th December 2022
- Create base version of project - User inputs a genre and a list of shots is outputted - Completed 13th January 2023
- Create shot statistics and output shot statistics – Completed 9th February 2023
- Create visual component - Completed 21th February 2023
- MVC implemented - Completed 28th February 2023
- Unity visual implementation done - Completed 2nd March 2023
- Unity Camera controls added - Completed 2nd March 2023
- + Add a card

Appendix B

The questionnaire used for user testing is outlined below.

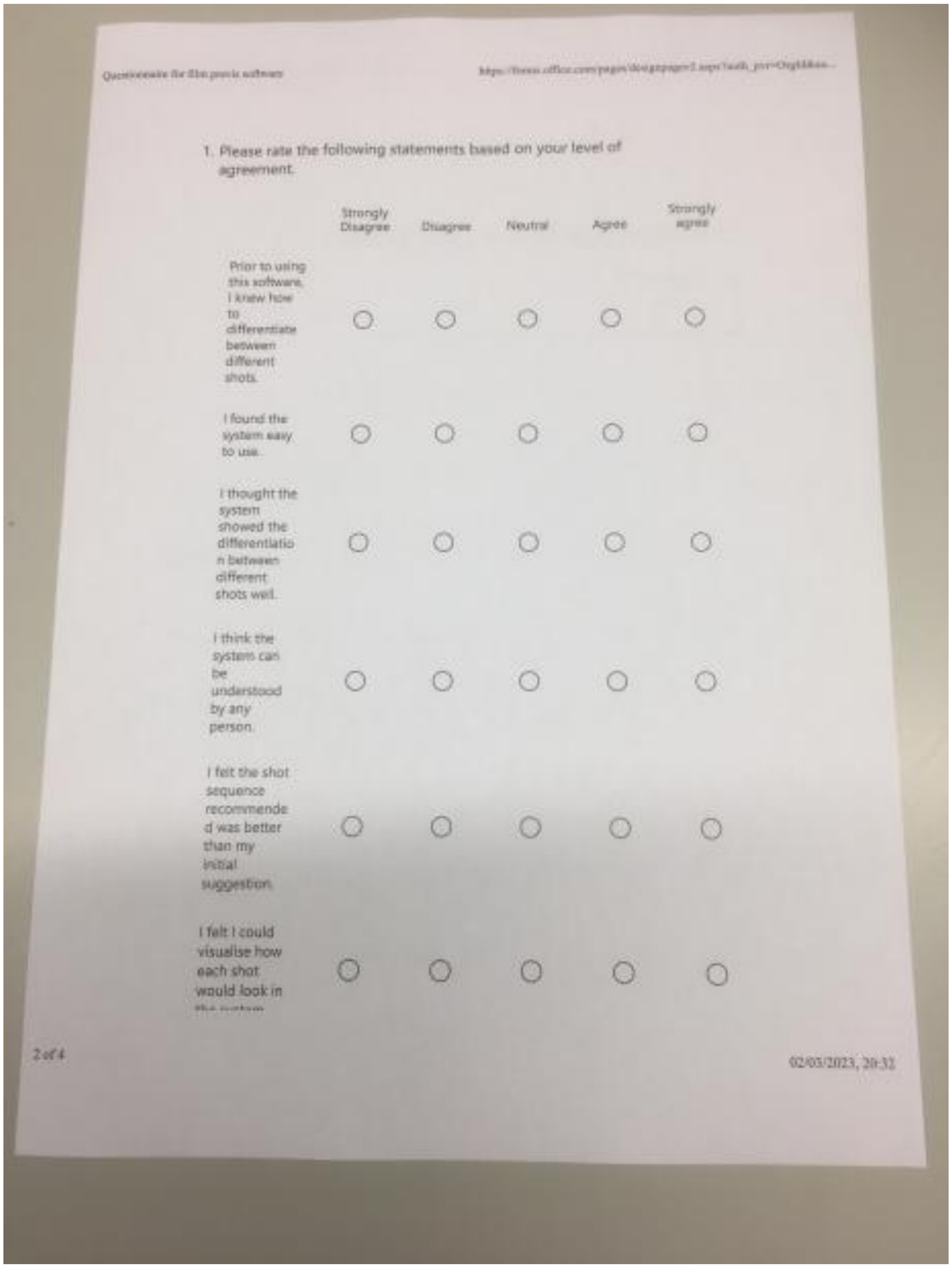


Questionnaire for film previs software   https://forms.office.com/pages/designpagev2.aspx?auth_pvr=OrgId&id=…

Questionnaire for film previs software

Please judge how much you agree with the following statements

1 of 4   02/03/2023, 20:32

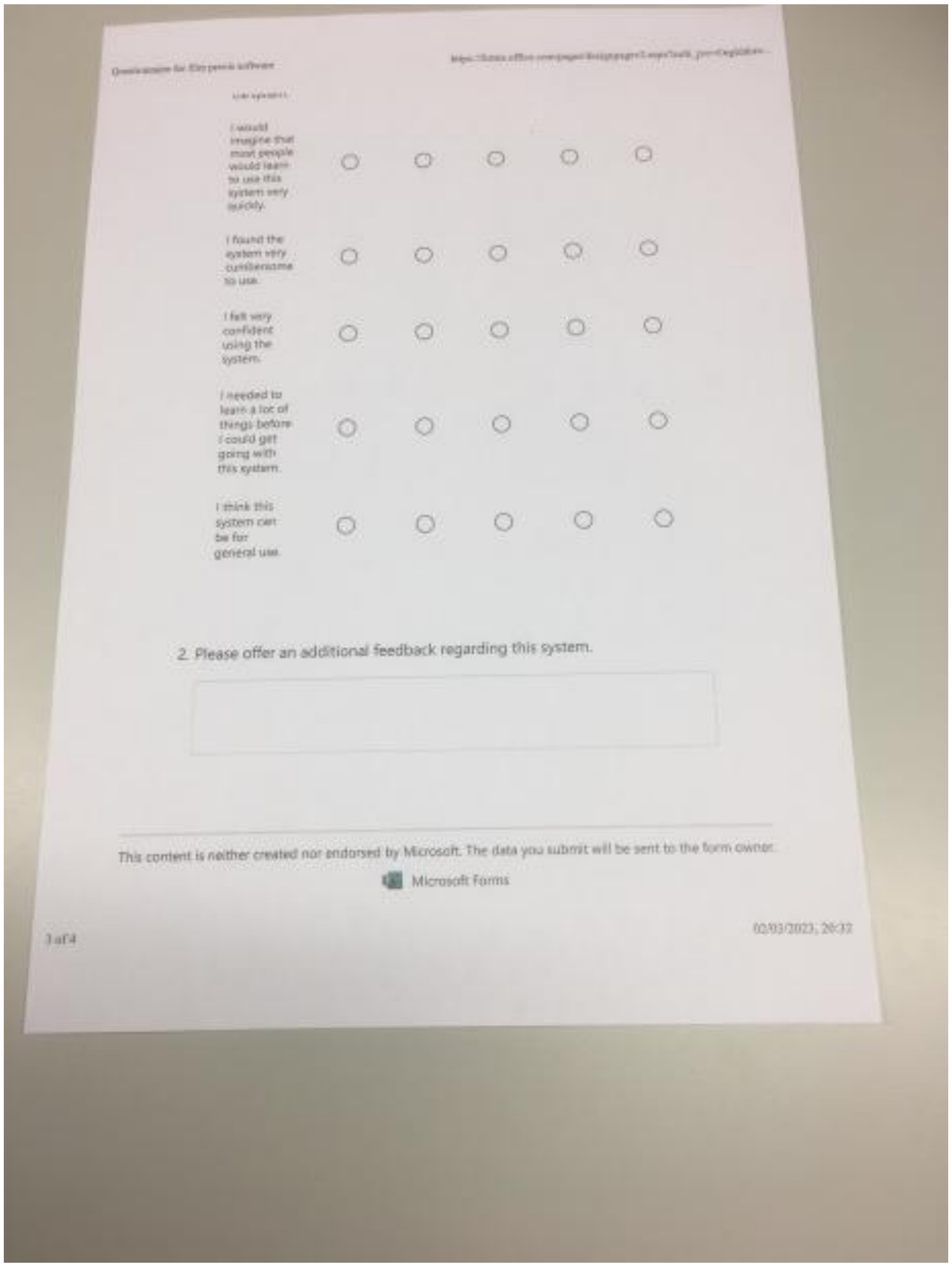


Questionnaire for film previs software   https://forms.office.com/pages/designpagev2.aspx?auth_pvr=OrgId&lan...

1. Please rate the following statements based on your level of agreement.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly agree
Prior to using this software, I knew how to differentiate between different shots.	○	○	○	○	○
I found the system easy to use.	○	○	○	○	○
I thought the system showed the differentiation between different shots well.	○	○	○	○	○
I think the system can be understood by any person.	○	○	○	○	○
I felt the shot sequence recommended was better than my initial suggestion.	○	○	○	○	○
I felt I could visualise how each shot would look in	○	○	○	○	○

2 of 4   02/03/2023, 20:32



Questionnaire for [illegible] software [illegible]

[illegible]

I would imagine that most people would learn to use this system very quickly.	○	○	○	○	○
I found the system very cumbersome to use.	○	○	○	○	○
I felt very confident using the system.	○	○	○	○	○
I needed to learn a lot of things before I could get going with this system.	○	○	○	○	○
I think this system can be for general use.	○	○	○	○	○

2. Please offer an additional feedback regarding this system.

This content is neither created nor endorsed by Microsoft. The data you submit will be sent to the form owner.

Microsoft Forms

3 of 4

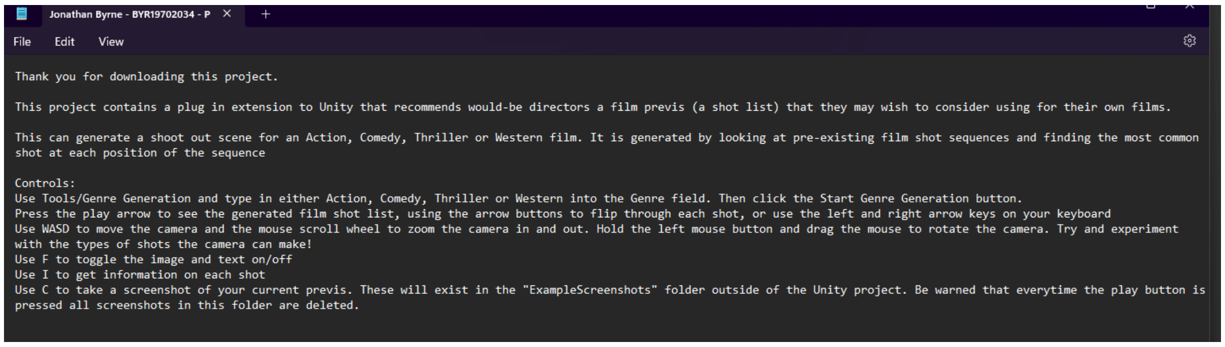
02/03/2023, 20:32

Appendix C

A rough outline of the project UI is detailed below:

Appendix C.1

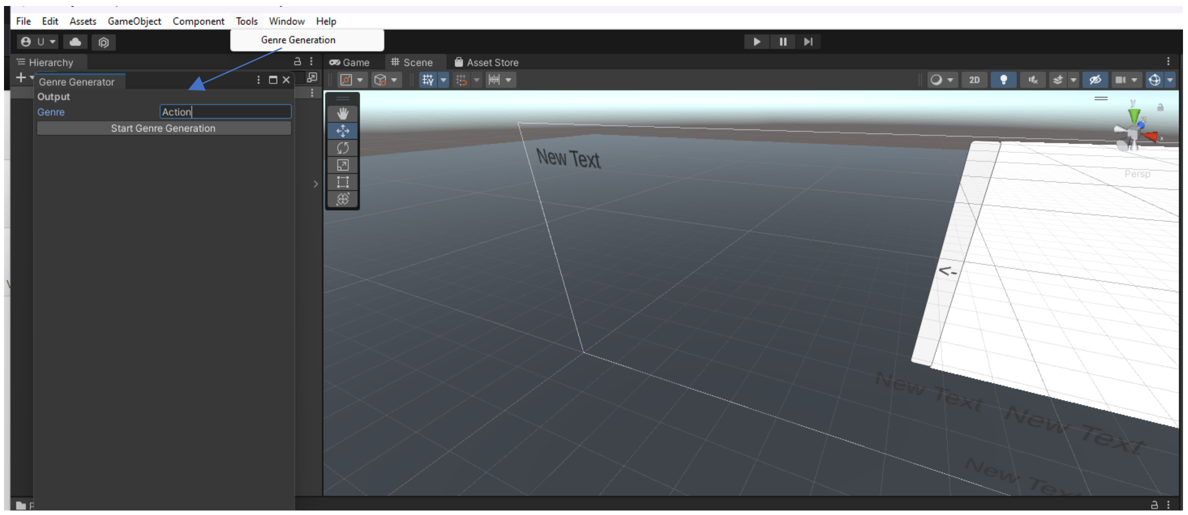
A read me document is included to give instructions to the user



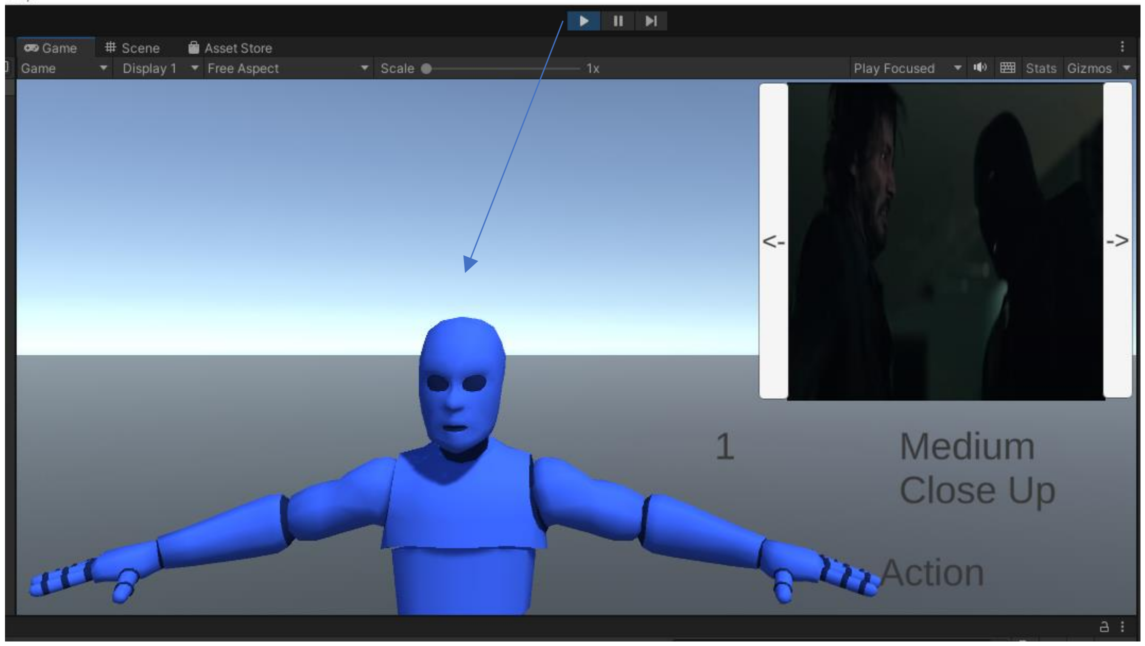
```
Thank you for downloading this project.

This project contains a plug in extension to Unity that recommends would-be directors a film previs (a shot list) that they may wish to consider using for their own films.

This can generate a shoot out scene for an Action, Comedy, Thriller or Western film. It is generated by looking at pre-existing film shot sequences and finding the most common
shot at each position of the sequence

Controls:
Use Tools/Genre Generation and type in either Action, Comedy, Thriller or Western into the Genre field. Then click the Start Genre Generation button.
Press the play arrow to see the generated film shot list, using the arrow buttons to flip through each shot, or use the left and right arrow keys on your keyboard
Use WASD to move the camera and the mouse scroll wheel to zoom the camera in and out. Hold the left mouse button and drag the mouse to rotate the camera. Try and experiment
with the types of shots the camera can make!
Use F to toggle the image and text on/off
Use I to get information on each shot
Use C to take a screenshot of your current previs. These will exist in the "ExampleScreenshots" folder outside of the Unity project. Be warned that everytime the play button is
pressed all screenshots in this folder are deleted.
```

Appendix C.2

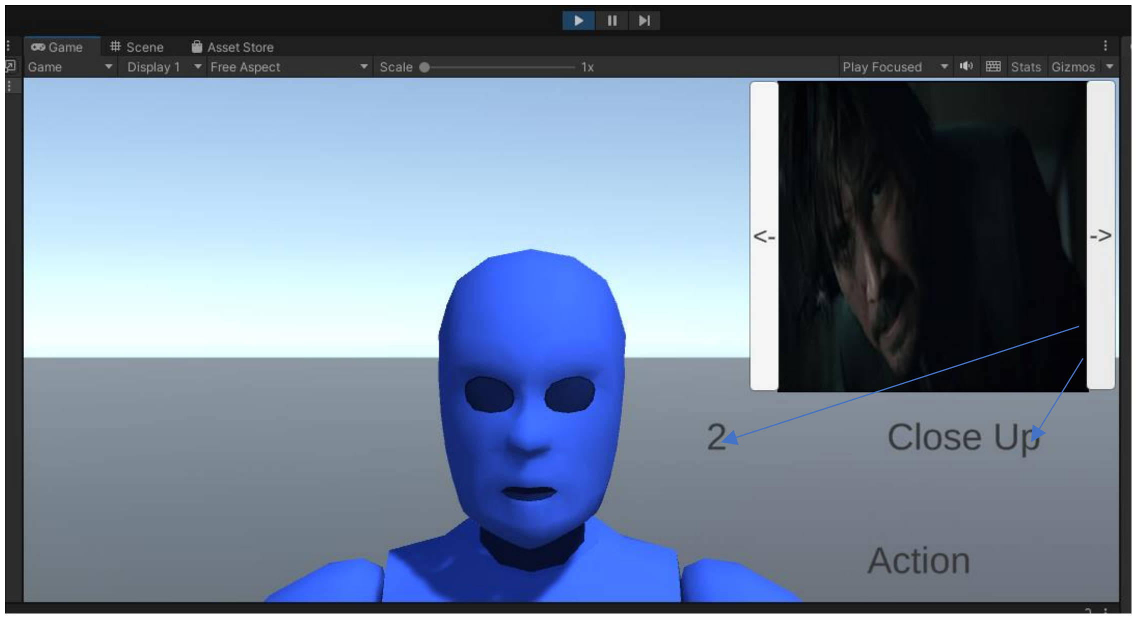
Following this guidance, C.2 shows a step by step work through on using this project.



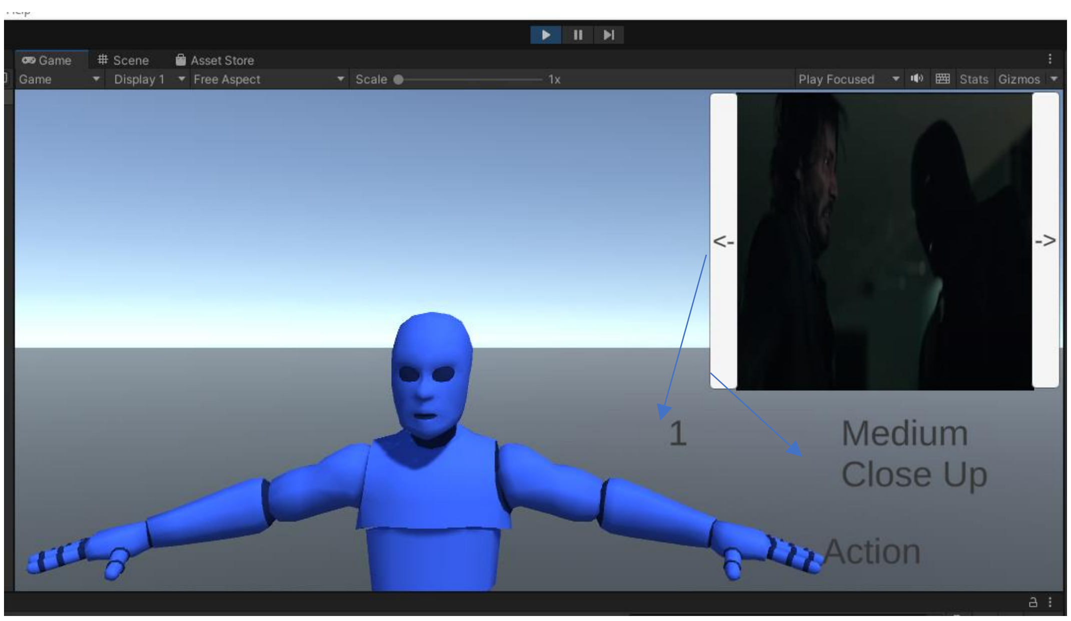
Firstly, the user would click the Tools/Genre Generation, type the required genre, then press the Start Genre Generation button.



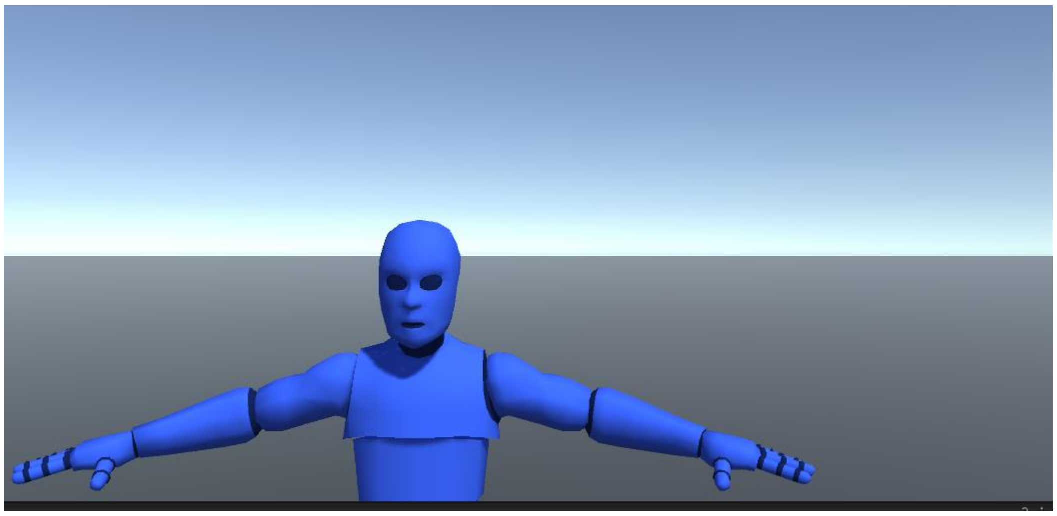
When the user presses play mode, they will have a shot list generated.



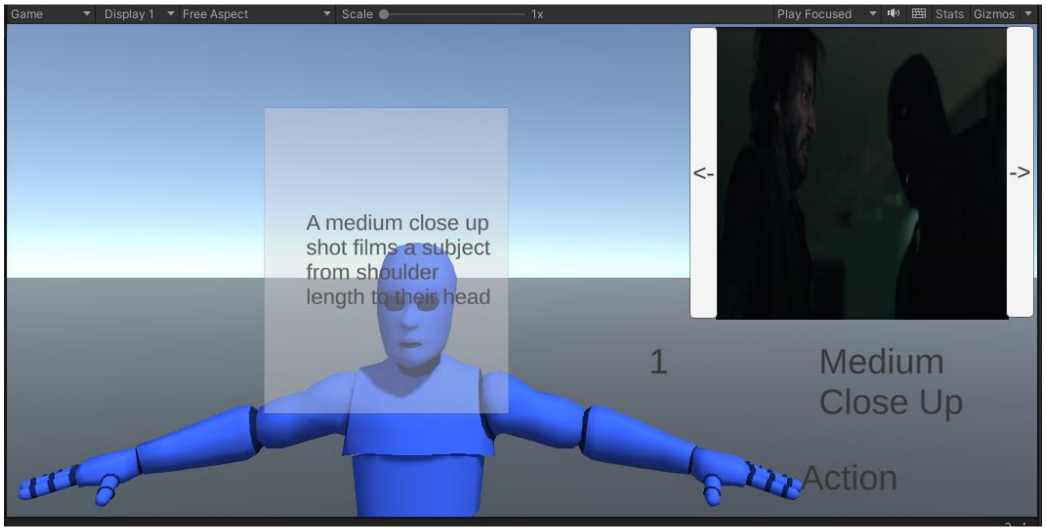
Pressing the right button (or pressing the right arrow key) moves the shot list forward by one.



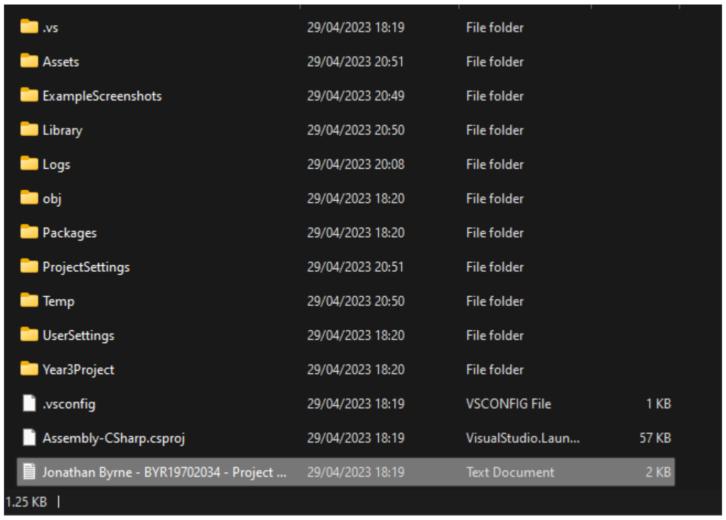
Pressing the left arrow (or the left arrow key) moves the shot list back by one.



Pressing F toggles the canvas on and off.

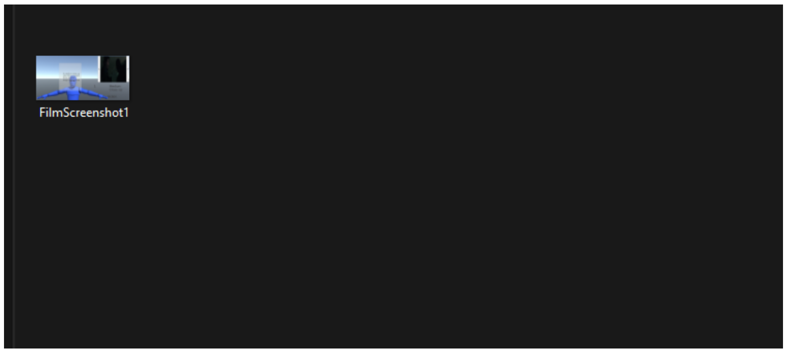


Pressing I toggles information about each shot.

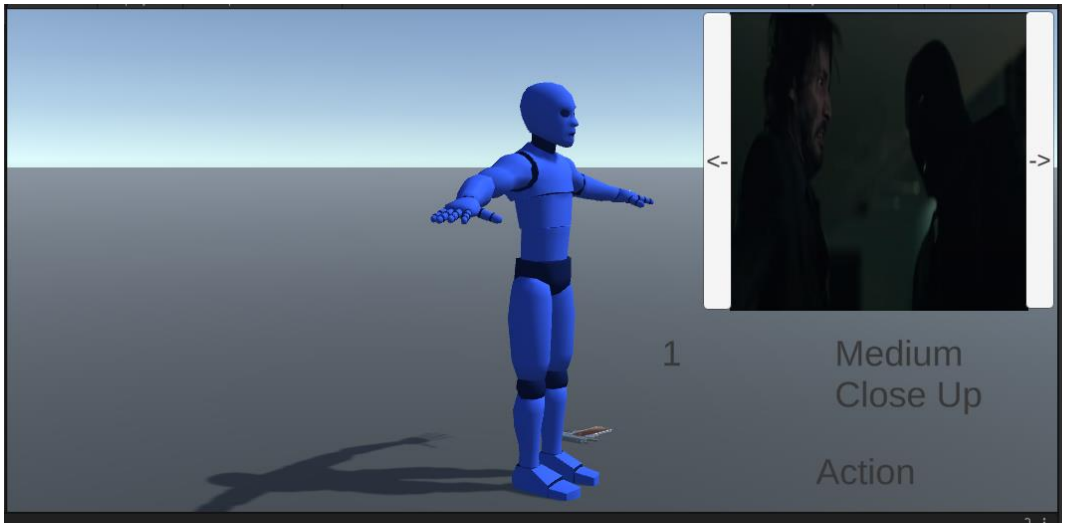


.vs	29/04/2023 18:19	File folder	
Assets	29/04/2023 20:51	File folder	
ExampleScreenshots	29/04/2023 20:49	File folder	
Library	29/04/2023 20:50	File folder	
Logs	29/04/2023 20:08	File folder	
obj	29/04/2023 18:20	File folder	
Packages	29/04/2023 18:20	File folder	
ProjectSettings	29/04/2023 20:51	File folder	
Temp	29/04/2023 20:50	File folder	
UserSettings	29/04/2023 18:20	File folder	
Year3Project	29/04/2023 18:20	File folder	
.vsconfig	29/04/2023 18:19	VSCONFIG File	1 KB
Assembly-CSharp.csproj	29/04/2023 18:19	VisualStudio.Laun...	57 KB
Jonathan Byrne - BYR19702034 - Project ...	29/04/2023 18:19	Text Document	2 KB

1.25 KB



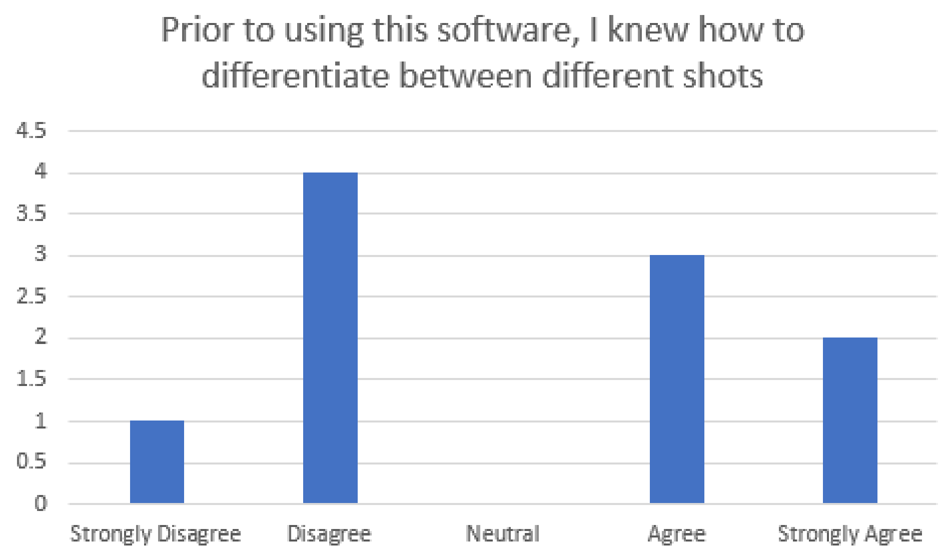
Pressing C creates screenshots in the "ExampleScreenshots" folder.

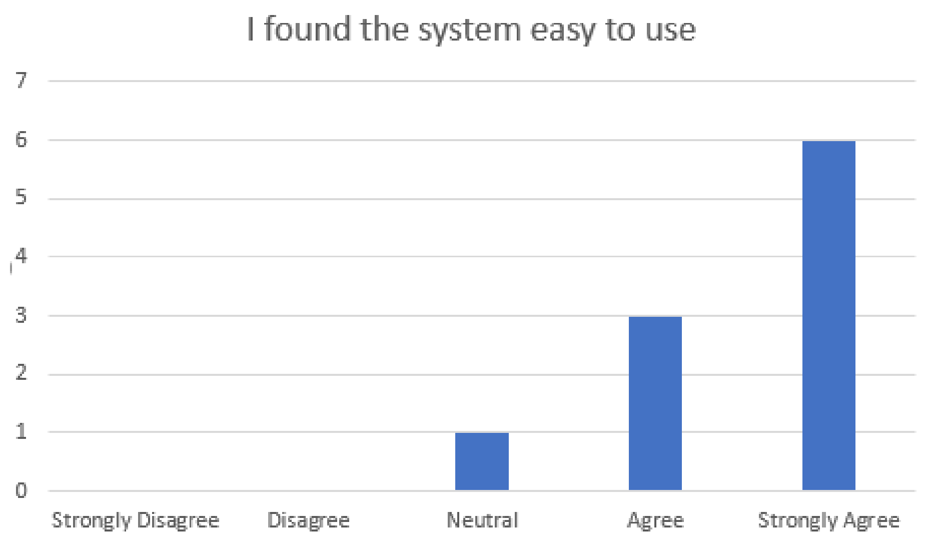


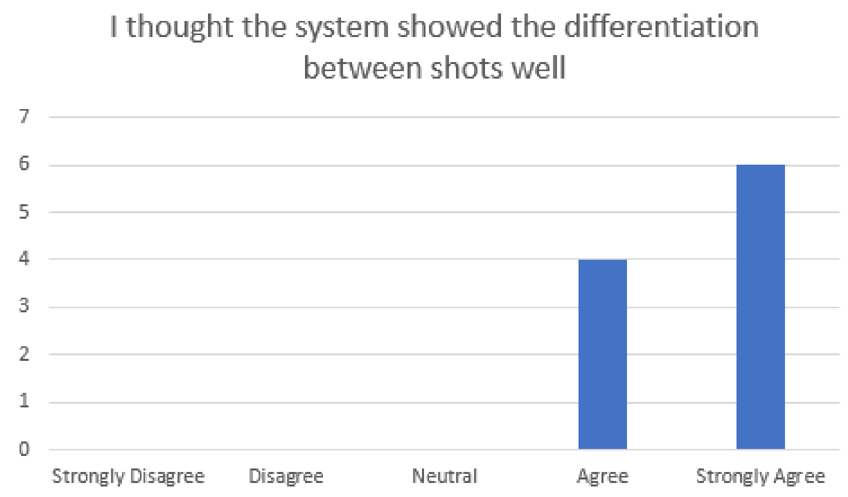
The user can also move the camera by moving it with WASD, using the scroll wheel to zoom in and out and dragging the mouse to rotate the camera.

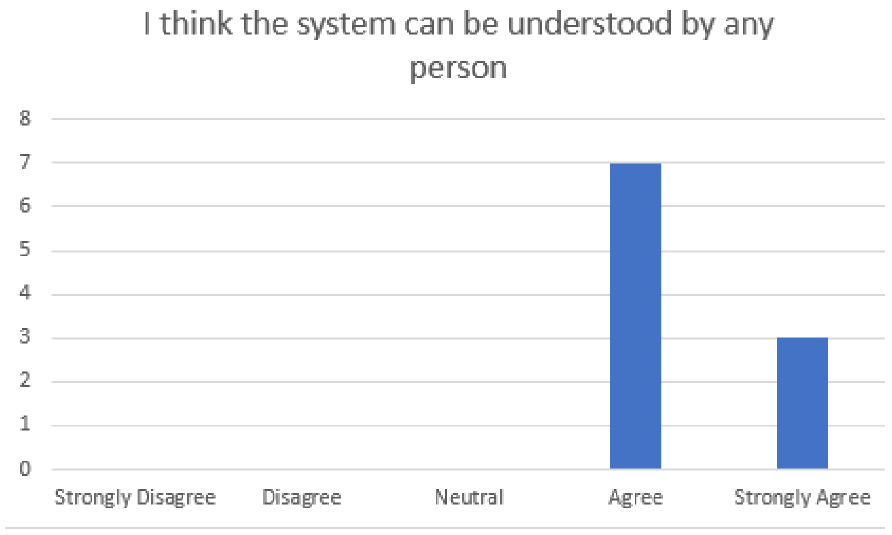
Appendix D

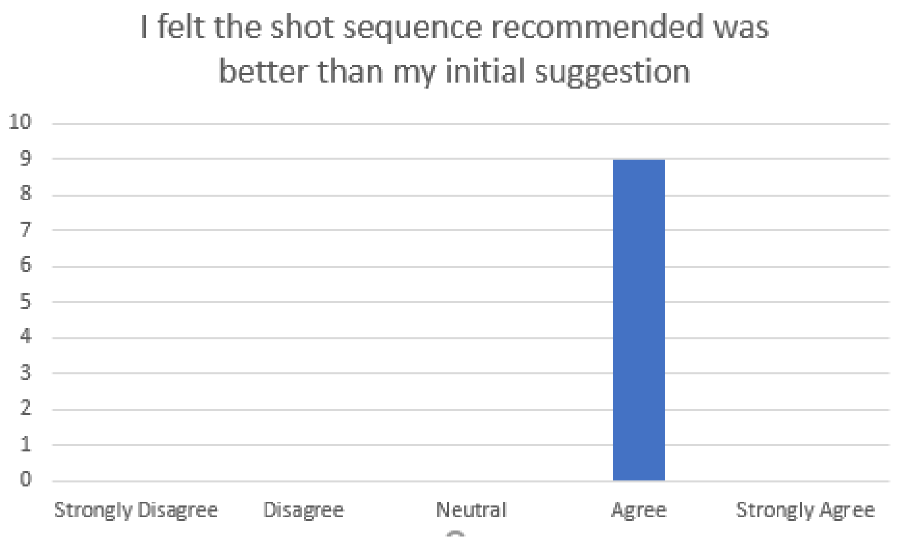
These are all participant responses to the questionnaire in Appendix B.

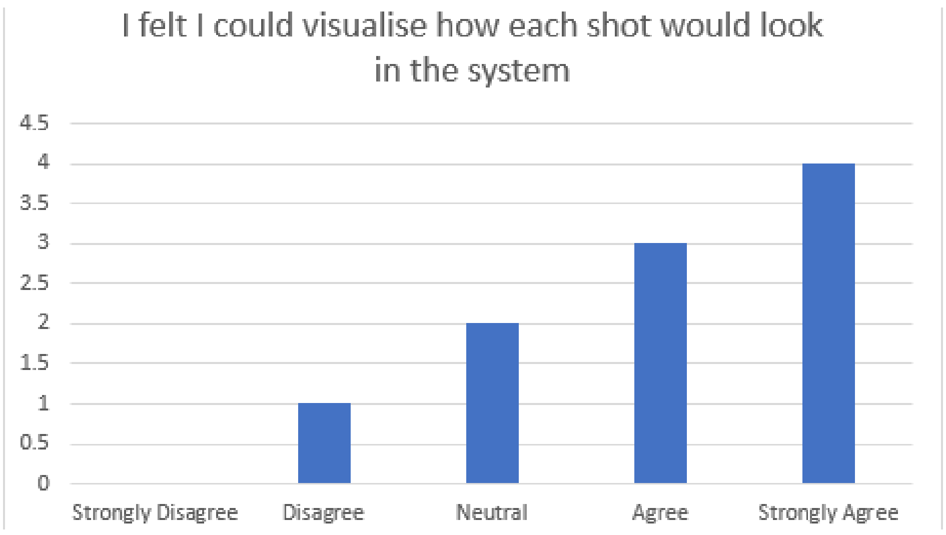


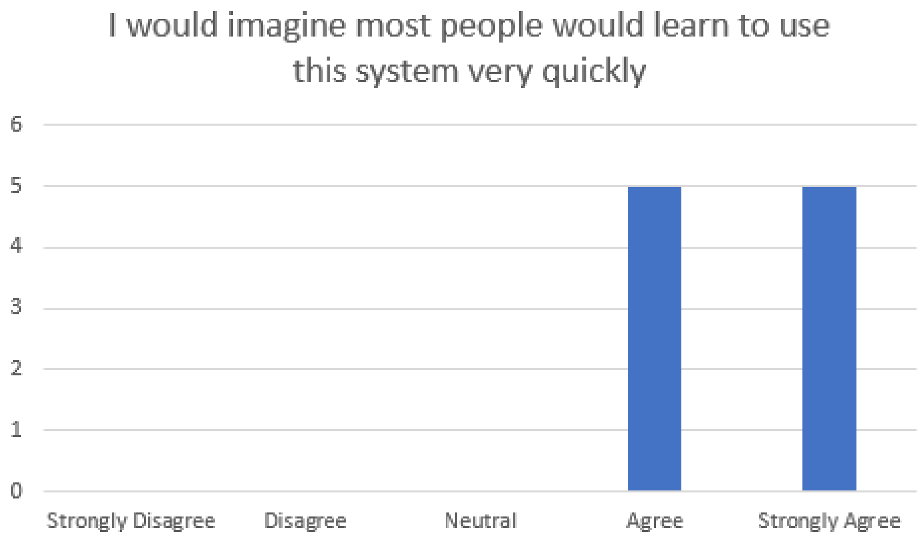


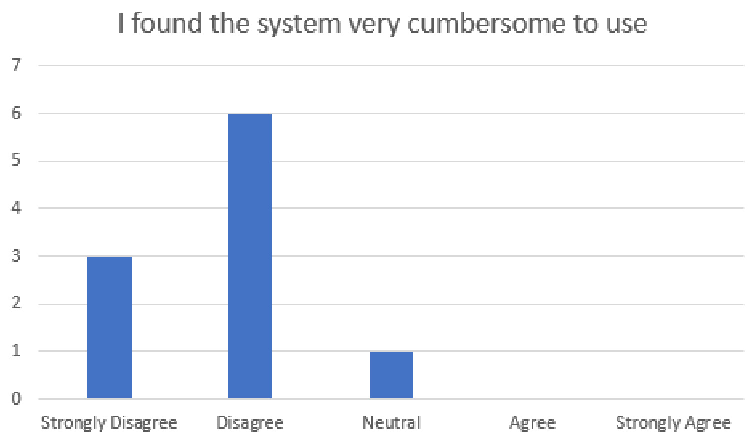

I thought the system showed the differentiation between shots well
7
6
5
4
3
2
1
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree

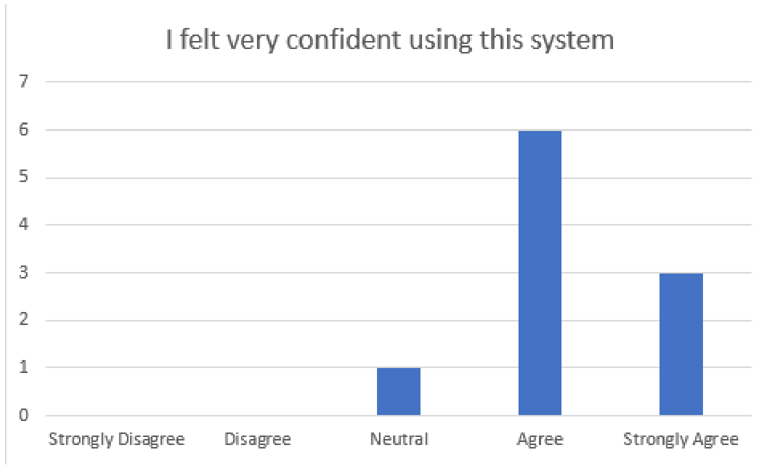

I think the system can be understood by any person
8
7
6
5
4
3
2
1
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree

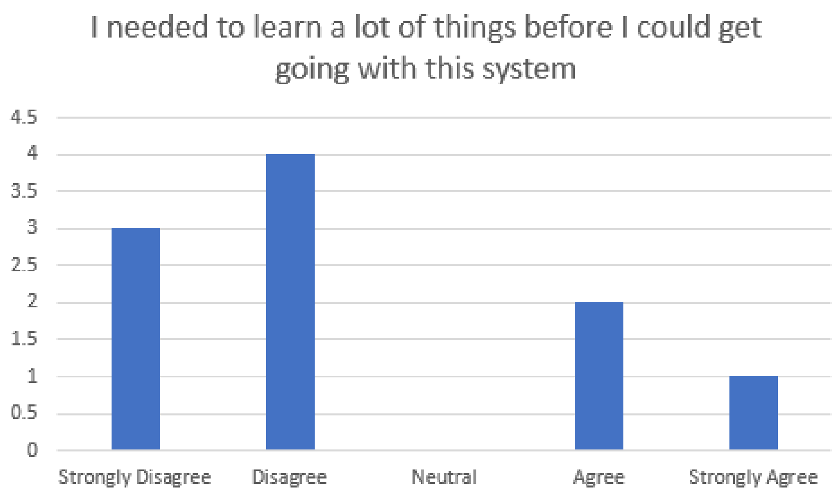

I felt the shot sequence recommended was better than my initial suggestion
10
9
8
7
6
5
4
3
2
1
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree

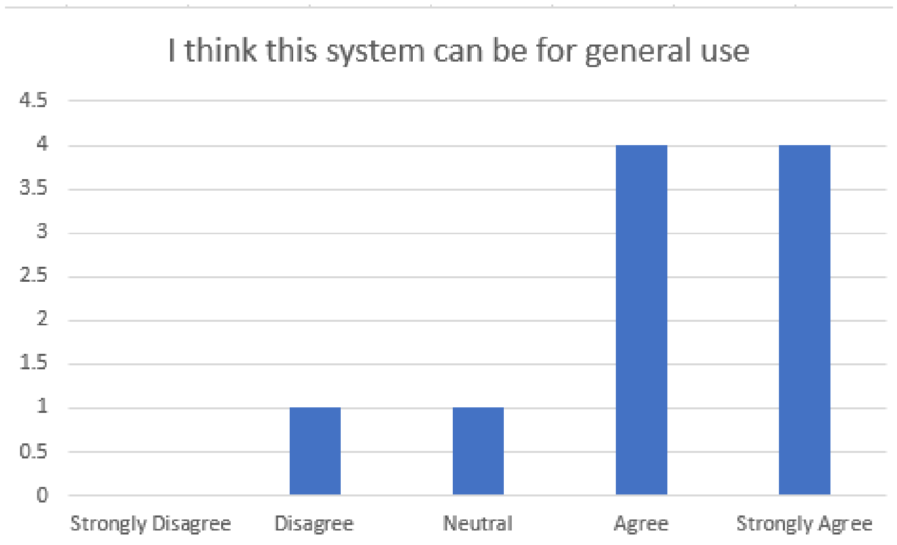

I felt I could visualise how each shot would look in the system
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree


I would imagine most people would learn to use this system very quickly
6
5
4
3
2
1
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree


I found the system very cumbersome to use
7
6
5
4
3
2
1
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree


I felt very confident using this system
7
6
5
4
3
2
1
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree


I needed to learn a lot of things before I could get going with this system
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree


I think this system can be for general use
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Strongly Disagree
Disagree
Neutral
Agree
Strongly Agree