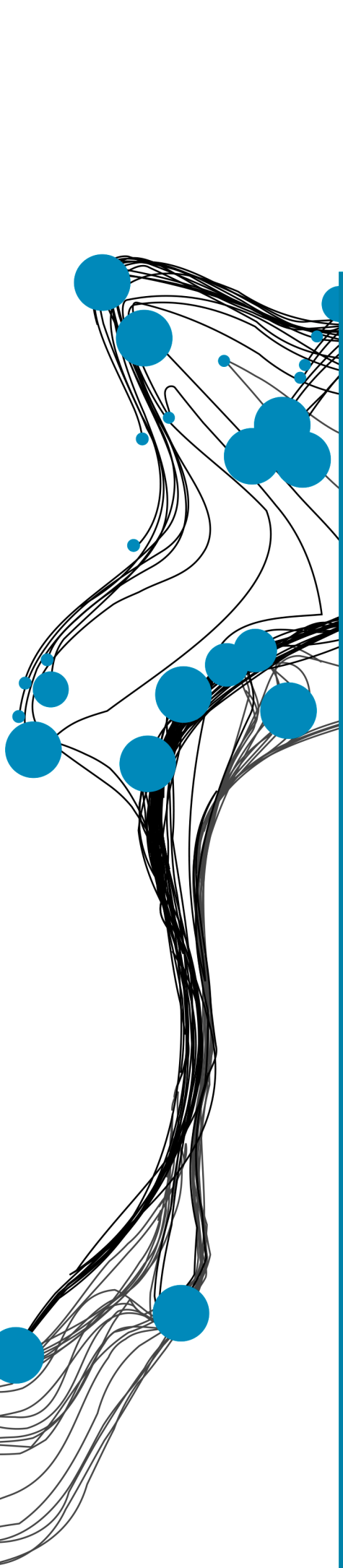


AutoPlay

Computer Generated Theatre

Design project - Technical Computer Science

Group 9:

Bart Batenburg

Denise den Hartog

Steven Tazelaar

Tijmen van de Meent

Supervisor

Dennis Reidsma

ABSTRACT

This report describes the process of creating the AutoPlay application, which has been commissioned by a graduating student of the Maastricht Institute of Perfoming Arts. The project was finished in 10 weeks, after which the final product was delivered to the client.

This project was part of the graduation process for the Bachelor study Technical Computer Science at University of Twente.

CONTENTS

1 INTRODUCTION

As part of the graduation process of Technical Computer Science, the Design Project serves as a means to show the academic problem-solving and project structure skills. This report shows one of such projects, and the process to finish that project, namely the process of the development of the AutoPlay application.

AutoPlay is made for the client to be used at Festival Cement. This is a Dutch theatre festival that takes place in Den Bosch. Our client is a student at the Maastricht Institute of performing arts and will be showcasing his graduation assignment at this festival. For him, we will facilitate an AI-generated theatre play.

2 INITIALIZATION

2.1 Design brief

Currently, a play is written by a scriptwriter which is then learned by heart by the performers. This results in the performers knowing the complete script while acting. Depending on the play, a lighting plan is also written. Herein is an overview of the scenes with a corresponding mapping of stage lights. Then during a play, the lighting technician is responsible for following this plan by controlling the lighting system manually.

In contrast to the described process, the product we deliver enables a new form of theatre. It can take a computer-generated theatre script, and allow for the performers to be acting the play live, without any rehearsals. A system was created so that the performers will know what to say and do at what time, such that they don't need to know the script by heart to play it out.

However, not every performer is able to act the entire play on the spot. Therefore a backup method is provided. It gives performers a chance to see their part of the script before going on stage. Of course, the main idea of the play is to improvise, so it is set up in such a way that a performer won't be able to rehearse the entire play.

Since the entire play is generated by a computer, the theatre lighting is also generated without the help of a lighting technician. Usually, this lighting technician would adapt the lighting based on what is happening on the stage. With the final application, the computer will take this place.

2.2 Plan of approach

2.2.1 Strategy

Our strategy for solving this problem was based on a phased out project approach. These phases enabled a more thorough focus on all specific steps that had to be done for the project to succeed. We have divided our project into the following phases:

1. Initialization
   In this phase the client's problem was established and an approach was created that would tackle the problem.
2. Definition
   In this phase all requirements were gathered and a solution was proposed to the client. This process is elaborated on further in chapter 3.
3. Design
   In this phase a technical design for the solution was made. This process is elaborated on further in chapter 4.
4. Implementation
   In this phase the proposed and designed solution was implemented. This process is

elaborated on further in chapter 5.

5. Testing
   In this phase the implementation was tested. This happened parallel to the implementation phase. This process is elaborated on further in chapter 6.
6. Validation
   In this phase the fulfillment of all requirements was validated with the client. This process is elaborated on further in chapter 7.
7. Handover
   In this last phase the project was prepared for grading and handed over to the client. This process is elaborated on further in chapter 8.

2.2.2 Planning

The first two weeks were spent on the initialization phase. This phase intertwined with the second phase, the definition phase. After these two phases, we received approval from our client, signifying that we were on the same page with regard to the expectations of the project.

These two phases together could be called a Proposal phase, since both the initialization and definition phase reports comprised the project proposal. When the proposal was confirmed, we moved on to the design, implementation, and testing part of the project. These phases merged into each other, meaning that they were not completely done before the next phase started in parallel.

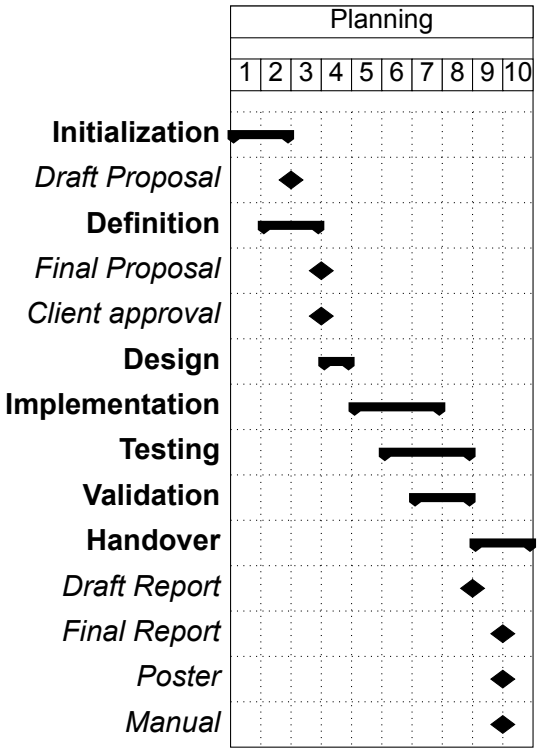


Figure 2.1: Planning in Initialization Phase

2.2.3 Deliverables

There were no set deadlines for this design project, except for a set date for the final poster presentation. We set internal deadlines for the following deliverables as explained below.

14-09-2021	Draft proposal
24-09-2021	Final proposal
22-10-2021	Draft report
01-11-2021	Presentation slides
03-11-2021	Final report
08-11-2021	Poster
12-11-2021	System & Manual

Proposal

The project proposal, was created in order to ensure that the expectations of client, supervisor, and project group are aligned. The proposal consisted of the two documents created in the initialization and the definition phase.

Presentation slides

In the final stage of the project a presentation needs to be given at the chair of our supervisor. The slides of this presentation will be handed in. The exact date of this presentation will still be determined.

Report

The report which lies before you explains the design choices that have been made during the process of designing the system. From the report it should be clear why the final product was designed as it is. Next to this, it includes the testing methods and results.

Poster

At the poster presentation on the 8th of November, all project groups participating in this edition of the Design Project presented their work of the past module. This was in the form of a poster presentation, which showcased the system we created.

System & Manual

The final goal of this project is to deliver the complete and working system together with a manual. The client should be able to set up the system and operate it without our help. Because of this, a manual will be created which clearly states all information needed to use the system.

2.2.4 Organization

This project was built up as an IT project with the waterfall structure as a mindset. Thus we created a plan to successfully deliver a product and mostly stuck to it. We also had meetings within the team and with the stakeholders regularly. We divided tasks between the team members based on the work they prefered and the workload they could handle. Additionally we reflected on the task division during the team meetings.

2.2.5 Roles and responsibilities

Almost all of us already did projects together and we knew each other for a few years. Because of this, we knew quite well where our strengths and weaknesses lie. Some of these strengths are that we were sure that everyone would put in the effort. No one was going to quit the module and disappear halfway, because we were all motivated to finish this project. Next to

this, everyone already had quite some experience in doing projects, both during their studies as well as outside of it.

Since we knew each other well, we also made some predictions about things that we would have to put a bit more effort and attention into. Moreover, we knew that our project group has no one who was an expert in UI/graphics design. This meant that we had to schedule more time to make everything look applealing and intuitive.

2.2.6 Workflow

Meetings

Every workday the team arrived by 10:00. We booked project rooms for every day we would gather with every team member. Progress was tracked on GitLab so that every member could view each other's progress. GitLab was also used to divide tasks over the team members, keeping in mind their strengths and weaknesses.

Meeting with supervisor

The team had one meeting the supervisor. The supervisor is a member of the HMI research group at the University of Twente. Their schedule was very full, so we tried to keep them up to date with our progress. Moreover, if any questions arised we would get in touch with them and ask for feedback and/or suggestions.

Meeting with client

Every week we had a meeting with the client of this project. This was planned according to their schedule. The goal of these meetings changed during the run of the project, depending on the phase of the project the meetings were used to gather requirements or serve more of a demo and testing role. Every meeting also contained a feedback session to make sure that the project requirements are fulfilled correctly.

2.2.7 Testing

To test the system we created a test plan. This plan included procedures for testing the system. This test plan is elaborated on further in chapter 7.

2.3 Risk analysis

In this section, special attention is given to the risks involved in this project. These risks are given a label: ‘extremely low’ to ‘critical’ according to the probability it occured and the impact the consequences would have had on the project. This would in turn indicate how much effort should be put into mitigating the risks.

The risks mentioned below are explained further in Appendix A.

Impact				
	High	*(medium risk)* Working at Home	*(high risk)* Illness of a team member	*(critical risk)*
	Medium	*(low risk)* No communication with the client Communication barrier	*(medium risk)* Underestimating labour involved Remote working without hardware	*(high risk)*
	Low	*(extreme low risk)*	*(low risk)*	*(medium risk)* Other obligations of team members
		Low	Medium	High
			Probability	

Table 2.1: Risk Analysis

3 DEFINITION

In this chapter the requirements of the product are explored, and the process to fulfill these requirements is defined. The chapter will describe how we gained insight into what needed to be done, how we made sure that was what the client wanted from us, and describe a process for us to work to efficiently create the system we defined. This chapter also includes a setup for the test plan that will be described later in this report.

3.1 Requirements

For the definition of the project, stating the requirements that will determine if the system is a good fit or not is a key step. These requirements, if described properly, gave insight into the components needed to develop and the methods that need to be implemented in these components. They also played an important role in the validation phase of this project, where we validated the end product of the implementation phase to see if the product adhered to the wishes of the client and the expectations of other stakeholders.

Requirements are generally split up into 2 categories, Functional requirements and quality requirements. The functional requirements state what the system should and shouldn't do. We have split them on priority: 'must have' requirements are crucial for the system to function properly, 'should have' requirements are not crucial but still desired while 'could have' requirements are only to be fulfilled if time allows. These requirements can be found in Appendix B.

And secondly, the quality requirements that state the conditions under which the system should perform the functions laid out in the functional requirements. These are for example the time it takes for actions to be completed and the number of inputs a system can handle. These requirements can be found in Appendix C and are laid out in terms of the main components we currently have defined for the system.

3.2 Proposed solution

During the play, the performers will need a way to receive the script to be able to know what to say and what to do. The proposed solution was to design an autocue system that will display lines and cues from the script. This autocue will be displayed on multiple screens positioned around the podium so that the performers are not bound to specific parts of the stage. The autocue also needed to be able to be advanced to coming lines and cues. This can be controlled by the performers.

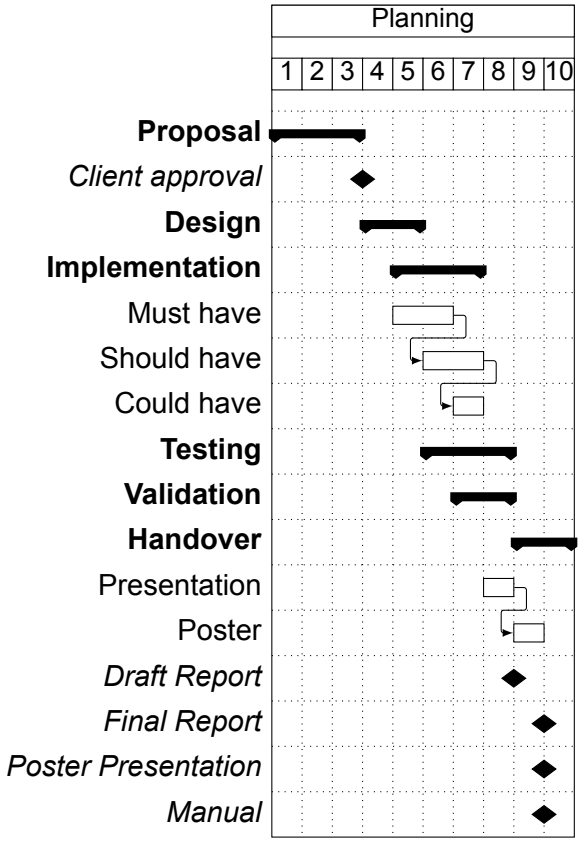
For the performers that are not able to do the entire play unrehearsed, an adapted script will already be given to them beforehand. This should however not contain the entire play so that the main storyline of the play will only be known to them when it is performed live on stage so they will still have to improvise. This adapted script should be automatically generated and should contain the lines and cues of the specific performers, including one line before and after it.

Finally, the stage should also be lit. Normally this is thought out before the play and then performed by a technician during the play. Using DMX (a professional lighting system protocol), our system will be able to manipulate the lights based on an algorithm.

3.3 Workflow

3.3.1 Planning

This schedule is an extension of the schedule in the initialization phase. We split up the implementation phase into the different requirement priorities. These overlap since some team members can for example already start working on ‘should have’ requirements while others are still finishing their last ‘must have’ requirements. Also, some time was planned to work on the deliverables such as the presentation and the poster.



3.3.2 Roles and responsibilities

Following the gathering of all the requirements, many new responsibilities emerged. Many of these can be categorized under certain roles that we have divided amongst ourselves.

The roles that were divided can be found in subsection 3.3.2, the division itself can be found in subsection 3.3.2.

Testing	Responsible for fulfilling the test plan
Back-end developer	Responsible for integrating all subsystems
Hardware	Responsible for anything hardware related
Editor	Responsible for reports and report layout
DMX interface	Responsible for designing and implementing the connection with the lighting system
Scrum master	Responsible for running the startup meetings and keeping track of tasks
USB interface	Responsible for designing and implementing the connection with the buttons the performers use
Lighting AI	Responsible for designing and implementing the system that decides how the lights are controlled
Poster design	Designing the final poster
UI design	Designing the user interface for the application
Contact person	Responsible for keeping contact with everyone outside of the project group
Autocue	Responsible for designing and implementing the autocue interface

We made the following responsibility division. This division is based on the end responsibility, but the person responsible will not be the only one working on this.

Bart:	Testing, Back-end developer, Hardware
Denise:	Poster design, UI design, Contact person, Autocue
Steven:	Scrum master, USB interface, Lighting AI
Tijmen:	Editor, DMX interface

4 DESIGN

Another major part of the project is the design of the application itself. This chapter serves to describe the technical design and note some requirements of the user interface. The technical design will be the description of components and how they work together. This phase also describes more clearly how the system will behave and how the interaction with the system will work. This was all in preparation for and as a result of the implementation phase.

4.1 User stories

There are two different types of people who will be interacting with the system. One of them is the administrator, who will be uploading the play, managing the settings, and starting the play. Next to this, there are the performers. They will be performing the play and advancing the autocue. We have created the following user stories for these two types of users, describing in short statements what they want to do with the system based on the previously discussed requirements.

Administrator

- As an administrator, I want to be able to start a play
- As an administrator, I want to be able to upload a script
- As an administrator, I want to be able to receive feedback when the script is uploaded
- As an administrator, I want to be able to create a backup script for an performer
- As an administrator, I want to be able to insert the light layout plan into the system
- As an administrator, I want to be able to adapt the look of the autocue
- As an administrator, I want to be able to abort a play

Performer

- As a performer, I want to be able to have a backup script
- As a performer, I want to be able to see the upcoming lines from the stage while performing
- As a performer, I want to be able to advance the lines on the autocue

4.2 Diagrams

To help aid design, the user stories can be turned into a series of diagrams that further go into the technical details of the system. These diagrams are made in a program called Visual Paradigm, that makes it easy to visualize the system. We have decided to make 3 diagrams in order to get our system requirements clear; an activity diagram, a use case diagram, and a class diagram. All of these will be explained below, and are attached in Appendix D.

4.2.1 Activity diagram

In Figure D.1 the activity diagram for this project can be found. This diagram maps out the way the user wants to interact with the system and thus gives information about the layout the application should have. This makes sure that the program provides a user with all the necessary information and control at each step and shows us where to return to if a step is completed.

4.2.2 Use case diagram

The use case diagram makes designers think about the different kinds of actors connected to the system. This information can then be used to make sure that these different kinds of actors get their requirements implemented into the system.

Since our program mainly focuses on the tasks the Administrator has to do, we will put our efforts into that element of the system. But we will also make sure that the system runs for the Performer by making sure that the autocue works. This will be described further in the test plan and the implementation details of the autocue element.

4.2.3 Class diagram

Finally there is a class diagram. This visualizes the connection between classes we would need to implement, such that the program will work correctly. The diagram gives good information for implementation, and thus significantly sped up that process.

4.3 System design

4.3.1 Elements

From the diagrams we have derived the key elements of our system, and what these should do. This is crucial to the system design because it gives insight into which classes to use, what we can look for in libraries, and what kind of communication between classes will have to take place.

Our first element is script importing and processing. This element will have to have some input for a script in a certain layout. Then, it should process the script by separating characters and storing their lines and making that information available for the other elements.

Next up is the backup script element. This will let the user pick a role, and store their lines in a separate file. Another element will be the settings page. Here you can set up the autocue to your liking, by changing settings like letter size and background color.

An important element is the autocue. This element will be used by the performers that will act out the play according to the script. The performers will provide input to the system. This is done by some sort of controller, which sends events to advance the autocue or roll it back.

And finally, there is the lighting plan and DMX control. This element will allow the administrator to input the lamps, their channels, the channel usage, and input the location of the lamps. This information will then be used to control the lighting. The lights will be controlled based on the lines of the play, with some form of emotion detection or a different text rating. The mood of the text will be integrated into the color of the lamps. These colors will be pushed over DMX to the lamps with the use of an Enttec DMX USB pro-mk2, which the client provided to the team.

4.3.2 Framework

In order to design the system from the abstract diagrams we had to pick a framework to build the system on, and study the way the framework works. Since we did not have much knowledge about frameworks for desktop applications, we searched for a system using an already known technology to render a desktop application. This system also had to be capable of controlling DMX, reading or writing files and had some kind of separation between back-end and front-end.

The framework we ended up deciding to use is Electron, an open-source framework developed and maintained This was all because the supervisor we have for this project suggested that we would work in this way. by GitHub. Electron allows for the use of web technologies like HTML, CSS, and Javascript for the development of desktop applications. It does so by running a background main process and a separate render process for each window, both with possible Node.js integration, and packaging a chromium rendering engine to render the front-end views.

This causes some significant overhead compared to running on more bare-bones C++ or C# code, but gives back a lot of support from web developers and their node packages, and ease in programming because of the simplicity of HTML.

4.3.3 Communication and libraries

The system needs some communication between the render and main processes. Luckily this is available in Electron through a package called IPC (Inter-Process Communication), which asynchronously handles and sends calls from one process to another. The IPC works by emitting events to user-defined channels, possibly with data attached to it. Multiple processes can listen to that channel, and consequently handle the event, use the data, and even reply to the process that triggered the event.

The system also needs to communicate with the DMX controller attached to the computer that runs the system through a USB interface. From testing, we found out that the DMX controller attaches through a serial communication port to the system and that this is easily accessible through the serial port node package.

For the DMX communication, we have chosen the node-dmx package, which contains all the software that handles the communication. Our code only has to point to the device location, tell the package what kind of DMX controller is connected, and tell it to set up the DMX universe. The lamps can then easily be commanded with an update to the universe from a set specified in the call.

4.3.4 Final system design

All these considerations lead to an overall system design. This has front-end render classes that handle all the manipulation of the looks and functionality of the pages of the application, and throws events based on user input. Next to this, it has the back-end thread that handles events, communicates with the system through electron, with the DMX lamps through node-DMX, and processes pdf files through pdf.js-extract.

A flow will be programmed that helps the user in running the system, which is based on the activity diagram. The user goes from script input, to configuring all the settings, to an autocue with light control handled in the background. This is also explained in the manual so that the user of the application knows what he can do at each step, and what to expect to happen next.

4.4 User interface

With a good system design that meets all functional requirements, the application should be working correctly. However, the interface design requires some steps to work through them.

The most important part of the user interface is using intuitive steps to guide the user around. The activity diagram in appendix D.1 is a basis for the steps the user has to take in the application. However, we chose to allow the user to always retrace their steps and, for example, upload a new script. This should not break the application. Next to this, the design should be simple and easy to work with. Complex menus or navigation hinders usability and should be avoided.

The implementation of the user interface will be discussed further in the next section.

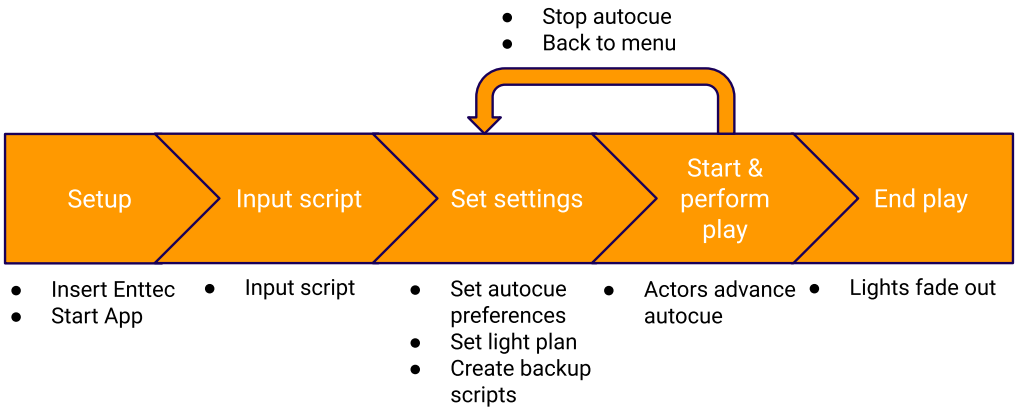


Figure 4.1: Application Flow

5 IMPLEMENTATION

In this chapter, we followed all decisions in chapter 4 to start building a functioning application to test and deliver in the upcoming phases. This phase ran slightly parallel with the design phase, as there were decisions for the design of the system we made while the implementation went on. We will describe how we built our system according to the elements we identified and then talk about the integration of those elements.

5.1 Script extraction

The extraction and parsing of a script are split up into 2 separate steps:

1. Read the script from a file into an Intermediate Representation (IR)

2. Parse the IR into a custom 'Play' object

This separation of steps is in place so that different file types can easily be added, without having to change the parsing step.

5.1.1 Read the script into an Intermediate Representation (IR)

Before the system can continue to the play configuration, the user will be prompted for a script. They can either choose to upload a PDF file (.pdf) or a plain text file (.txt). When the user confirms their choice, the system will select the corresponding reader.

The intermediate representation contains information on every piece of text that can be received from a reader. Per piece of text, it should contain the height (font size) and width (corresponding to the amount of characters) of the text on the original page, as well as the text itself. The height is used to merge pieces of text that were originally on the same line, the width is used for possible expansion of the play parser to differentiate between cues and speech lines.

TXT

Plain text files are read line by line, and the height and width are set accordingly. These files don't need more processing, and can natively be read without any special libraries.

PDF

PDF files are read using the pdf.js-extract library. This library uses pdf.js, a javascript pdf viewer for browsers, to read PDF documents page by page. Per page, it outputs a bulk of data, which is processed to the intermediate representation.

5.1.2 Parse the intermediate representation

The parsing step merges all pieces of text that are on the same line in the original script. After this first step parsing continues by taking the first non-empty line from the script and setting it as the title of the play.

For all next lines until the end of the script, it will try to find line assignments (lines that end with a colon) and parse the roles for the following lines. This is done by splitting up the part in front of the colon at strategic positions (commas, ampersands, the words 'and' and 'or'). The lines that follow such an assignment are all bundled together to form a single piece of text, so the data object follows the structure of the script.

The resulting object can be exported to a readable format for the front-end of the application. This export results in a data object in which only relevant data for the front-end is present, to make processing on the front-end part minimal.

5.2 Backup generation

The object generated from parsing the input data can also be used to create a backup script. Following the requirements, this should be possible for a single performer while also displaying a specifiable amount of lines before and after the line of this performer. The application does this in 4 steps:

First, the application gets user input. They can choose a specific role for which to create a backup script, or generate them for all available roles. It is also possible to change the settings for the number of lines before and after. Finally, they can choose the filename and path for the resulting backup script.

These settings are then saved by using the normal method described in section 5.4. The Backup script generator then creates a long list of all the lines including the role associated with them. While doing this, the lines get parsed into the correct format to add to the final file. The creation of a separate object is done so this object can be modified without worrying about influencing the rest of the application.

After the creation of this new object the generator runs a filter where all lines that have the specified filter or are inside the range specified by the user get a flag. This is to make sure that lines are not added multiple times. If a line already has a flag then reading this flag would not change anything.

Finally all lines with the flag are added to a PDF file formatted such that the user can easily understand them. This is done by adding the role name and line number to each line and adding a separation when the specified role would not have to talk. An example is shown in **??**.

5.3 Autocue

After parsing the script, the autocue can be started. This opens a new window that requests the current settings. It extracts the needed information: text color, background color and the amount of lines to be displayed. The first two are applied immediately to the markup file. Then the autocue creates the amount of line objects according to the last setting. These objects get then populated with the corresponding content.

When a performer requests the next line an event is triggered and a request to the back end is made. The back end keeps track of the current location in the script and provides the front end with the now needed lines. These are then updated in the corresponding object. The process is the same for the previous line.

...
(721) VLADIMIR :
Unless they're not the same . . .
(722) BOY :
(off). Mister!
Estragon halts. Both look towards the voice.
(723) ESTRAGON :
Off we go again.
(724) VLADIMIR :
Approach, my child.
Enter Boy, timidly. He halts.
(725) BOY :
Mister Albert . . . ?
(726) VLADIMIR :
Yes.
...
(730) VLADIMIR :
What is it?
(731) BOY :
Mr. Godot . . .
(732) VLADIMIR :
Obviously . . . (Pause.) Approach.
...

Figure 5.1: Example of a backup script

5.4 System settings

The requirements also specify settings that should be changeable by the application itself. These are: autocue text color, autocue background color, autocue text size, backup role and backup range.

To make it easy to later add or settings, all settings are stored in an easily accessible object. Each front-end system can request these settings and extract the specific values needed. The user can change the settings on a separate page in the front-end view.

5.5 DMX lamp control

5.5.1 DMX

As per the requirements, the system needs to control lighting on stage through the DMX protocol. This protocol works by connecting the lamps to a universe. Such a universe has 512 channels, with each the possibility of having a value from 0-255 (one byte).

Each lamp can then be set to a starting channel, and have some amount of subsequent channels control an aspect of the lamp. A dimmer for example will dim its output according to the value sent by only 1 channel, whereas an RGB spotlight can have 4 channels for Red, Green, Blue, and White, or have even 20 channels where it binds channels together to get more precise control.

5.5.2 Lamp setup

For inputting what lamps need to be controlled, what kind of lamp they are, and where the lamp is located there is a page in the system where the user can add these lamps. This page contains a form for entering the start channel, the number of channels the lamp uses, and the category of the lamp. Javascript then adds a new part of the form in which the user inputs where certain parts of the lamp are, so for an RGBSpot it asks the Red, Green, and Blue channels as well as the often used master channel that dims all the colors evenly. The user can then press a button and add a new lamp.

After filling out the form and pressing the button, a square will be drawn on an HTML canvas with the starting channel printed in the square. These squares can be dragged around the canvas,

and the position the square is placed in will be taken as the location where the lamp is in.

To make the computation for the color of the lamps easier on the program, the theatre hall as modeled by the canvas is divided into 9 blocks, and each lamp which square is placed in a certain block is thus attributed to the block and controlled the same way as the other lamps in that block.

5.5.3 Controlling lamps

When the lamps are set up, it is time to start communication with the DMX controller attached to the device the program runs on. To program this we received a DMX controller from the client by post. This is an ENTTEC DMX usb pro mk2. The program is thus made for this controller, and tested as such, but may work for other controllers using the same ENTTEC pro chip.

5.5.4 Library

The first thing to find was a library to connect to the DMX device. For this, we found the Node-DMX library during the Design Phase. This library does most of the work and just needs a path and what model of controller. For finding the path to the controller the same module that the Node-DMX library uses internally for communication is used. Testing discovered that the ENTTEC connects using a COM port connection and those are easily listed with the serial port node module.

5.5.5 Path

The software first lists the serial port devices, finds an ENTTEC device, and then lets Node-DMX connect to that path, with the model hardcoded. The controller then begins to flash a green light on the controller telling us it is in control of the lamps and we can send it updates.

5.5.6 Sentiment analyzer

After that, when the autocue is started, each line triggers a few steps. First, the line itself is run through a sentiment analyzer. For this, it first takes out all the contracted words like “you’re” and turns them into their full form. Then it makes all words lowercase and tokenizes them. After that, the text is spell corrected and the sentiment is analyzed. This returns a value between -3 and +4 from negative to positive. This does currently only work on English text since the code is set up to correct English spelling and detect moods in English text only.

5.5.7 Lamp update

With this negative-positive rating, the system computes a color to give to the lights. This is done by some clever mathematics and generates a separate value for each block. If the emotion tends to be positive, the colors will be green, and if the emotion comes back negative, it will tend to be red. This is done by making the emotion value a significant weight in the calculation.

To make the colors on the set not similar to each other if for example a sentence repeats, there is a random component to all the color values each time an RGB set is requested. This will make for a very dynamic set of lights during the play. For the lamps that have only 1 channel, an average value is made such that the dimmers will have about the same amount of light as an RGBSpot in the same block.

The set of new values per block are then mapped on the previously configured lamps for that block. Each RGB channel will be set according to the RGB values, each dimmer to the averaged

dimmer value and all master channels are set to the same value in order to get the best block and emotion contrast. This is done for each block, such that all lamps that are configured should have new values to display. These new commands are then pushed to the node-DMX package, which updates the internal state of each channel to start broadcasting the new commands.

5.6 Integration

One of the last steps to take in the implementation phase is the integration step, combining all separate features into one working system. Some of this the work was taken out before we even started the implementation since the application design was decided on beforehand. Moreover, the integration steps for the different components were thought out beforehand, such that the implementation could take those into account.

In the integration, we made sure all components were accessible from the main interface of the application, made sure everything happened in the right order, and that all data was handed over correctly. This was often done by letting the main class run by the back end take care of storing the data and everything else to ask for data when they needed it.

5.7 User interface

The user interface should, as described in the Design section, be intuitive and easy to work with. To achieve this, we created a flow in the application that is represented in Figure 4.1.

5.7.1 Pages

The application includes 6 different pages, as well as the autocue window. These pages were connected and allowed seamless navigation between them. Each page had a function, corresponding to the application flow shown in Figure 4.1. The page views can be found in Appendix E.

5.7.2 Navigation logic

The page navigation is split up into a few different components. First of all, the front-end pages are all split up and showing one page removes the other. This way we guarantee that the application does not show multiple pages at the same time. Secondly, the navigation bar on top shows possible navigation steps the user might take. As you progress through the application, more navigation options appear. This ensures that the user cannot open a page if the requirements for that page are not met. Moreover, this navigation bar functions more as a reminder of pages that can be visited to revise settings or configurations than to be used very often.

Lastly, on the back-end, each navigation attempt is processed to check if the user can already open the corresponding page. For example, if the user wants to show the page for exporting backup scripts when they have not uploaded a script yet, the back-end won't allow the change of page and the navigation request won't be resolved. Additionally, the back-end serves data to the front-end if a page needs information to be pre-filled. This is used for the 'Upload script' page, for example, where the back-end serves the play that is at that moment loaded in if there is any.

5.7.3 Styling

The layout and styling make use of the Bulma (bulma.io) CSS framework. This ensures that the whole application has a single feel to it, and all elements tie in with each other. Moreover, it gives it a unique but familiar look. As stated above, the final designs are in Appendix E.

6 TESTING

A perfect project needs a good product, and to make sure the product that was developed is up to the standards, testing needs to take place. We made a plan in the definition phase for this. In this phase, we expanded on that Test plan, described some testing done during the development and the decisions they led to, described how the testing with the client was planned, and what the results were.

This testing is partially done during development, to see that the code you just programmed works, developers often log the outputs and confirm that that code did the right thing. One can also do integration testing to make sure that all elements of the system are working together in the right way. This is usually done by impersonating the user and using the project as intended. This can be done by hand for small projects, with larger projects usually using specific frameworks built for their codebase that performs the same actions as a user would in a repeatable manner.

6.1 Test plan

Early testing is essential for a good running project. Testing will make sure that the features that were built are properly implemented and don't contain bugs.It will also make sure that the product's development is heading the way the client specified.

6.1.1 System testing

Since there was not much time left for building unit tests into the system, the original plan, to build them for every separate feature and thus make the system test itself, was adjusted so we did more user testing than system testing. The adjusted plan was for everyone to take care of an element, as described in the design phase, and make sure during programming that that element behaved as it should.

We are using some hardware in this project, so we will need to test these physically. DMX512 is used for controlling the lights. Since we don't have an easy way to test the lights at the venue itself, we have contacted the university's theatre technology department and arranged to borrow a DMX-enabled RGB spot for some time to test if the connection to the controller works.

6.1.2 User testing

The user testing for our product will consist of two parts. Ourselves acting out the flow a user would take, and possible ways a user can deviate from that flow that could break the application. This would be done by running the application and using it as the user would. Besides that, the client of the product will be shown the current state of the application during meetings and thus be given the opportunity to give feedback on what we have built so far and steer the development in the direction of his desired product.

6.2 Test outcome

For this project, the team did a lot of testing which generated many different outcomes. In this section, we will describe some of the testings we did that we thought were worth mentioning, what the outcome of that testing was, and how we changed our design based on that.

6.2.1 System tests

Electron

Before we committed to go and program this application in the electron framework we first had to test that the framework was indeed a good choice for our situation. This was tested in a manner that was also useful for the further development of the application, by all performing the get started project. This confirmed our assumption that JavaScript was relatively easy to program in and had no quirks compared to normal web development we needed to deal with.

PDF extraction

For the extraction of script data from PDFs, we first needed to test what library was best to use to extract the text from the scripts. For this test, we needed an example script to process and verify that the system could understand the script properly. Since the client was not able to generate a script from the OpenAI system yet, we discussed what script would be similar to the output the client was expecting from the OpenAI system. We decided on the use of the script from “Waiting for Godot” by Samuel Beckett.

After a script was chosen, it was time to start out trying to import this script into the application. Testing different libraries we came to the conclusion that most data could be extracted with the pdfkit library and then building our representation of the play in JavaScript classes.

The previous solution worked well with text extraction, however, the PDF could contain more information, such as italic text. Using a different tool, pdf.js-extract, we tried to get that information. However, it happens to be that pdf files are not as uniform as one might think; there are multiple ways to specify italic text, which meant that only very complicated code could actually find the difference.

DMX

The DMX lighting system interfaces directly with hardware in both the communication with the controller and finally the output the lights generate. As explained in the test plan of this testing phase a lamp was borrowed from the university’s theatre department for testing. The controller we used for testing that lamp with our system was sent to us by the client, which purchased it for this project.

For testing the connection to the DMX controller, all that had to be done for set up was checking the serial ports for Enttec labeled devices, feeding that path to the node-DMX library, and starting up a universe in the DMX library. When this happened the LED on the controller started to change from flashing white to green continuously. From the manual, we read that this means that 1 of the 2 universes the controller is capable of controlling is connected and sending commands. The other universe we will not use.

For the testing of the lighting equipment, the library was set up to detect an Enttec connected to a device and connect to it just as in the previous step. But this time there was also a lamp connected to that controller. At the same time as the controller started flashing green light to us, the lamp started to show a red LED to confirm that it was receiving DMX commands.

A function was then programmed that used a button to set some channels to a random value. We found out that the lamp was using channels 1-3 for R, G, and B and channel 8 for Master. Channels 4-7 were used for simple color control, strobe speed, color switcher, and color wheel. All this simply confirmed that the control of lamps using this method was working, and thus this is what we generalized to the control of all lamps.

Since the DMX element was programmed on Linux, we had to check that the system would still work if used on different operating systems, because they sometimes handle hardware connections differently. After trying to connect the DMX controller to windows we found out that the serial port library listed some different outputs for the DMX controller on windows. Thus we needed to change how we check for an Enttec device in the program by not using the manufacturer name and using the serial-number instead.

Pointers

The pointers we use for the control of the autocue by the performers are generic presentation pointers. These devices show up to the system as a human interface device and the forward and back buttons of these pointers are mapped to a press of the PgUp and PgDn buttons of a normal keyboard.

Since these button clicks are listened to as events, we had to test that a possible collision would not break the program in any dramatic way. For this 2 pointers were purchased. Testing was then done by simultaneously press buttons on both pointers to see how much the system could handle.

Testing showed that when multiple pointers were pressed to the next line at the same time, the system registered those and the Autocue advanced by two lines. This will be discussed in a later section of this report as there is a point of improvement possible here that was not implemented.

Another thing testing showed that when 1 pointer presses back and the other forward the system glitches between the 2 commands a bit and ends up back at the same line that was already on the screen. This behaviour was expected.

A final point of interest was that the autocue sometimes had some trouble keeping up when scrolling through the autocue was done very fast. The back-end probably has some inefficiencies that cause the loading of the lines to take some time, besides the fact that the entire AI is called with each line change. This could be improved upon, but at the speed the testing was going to cause this the autocue would be unusable for performing anyway, so this is not a real problem that needs to be dealt with.

Mood detection and lamp control

For testing the mood detection we use and the math we use after that, we programmed the system and then could feed it lines we entered and look at the output. Both the generated emotion score, as the final values that were pushed to the lamps were checked to see if they matched the expectation and conformed to the requirements.

Backup script generation

The backup scripts are generated according to the user settings. These determine what character to generate the script for and then how many lines before or after each line of that character you want to include in that backup script.

To test if this backup script generation doesn't go wrong, we tested entering big values for the amount of backup lines. In a version of the code we had a lot of repetition of lines which was not efficient. Then the system was changed to not put the same line in the backup script multiple times, which means that generating the backup script with a large number of previous or next lines just ends up copying the entire script.

6.2.2 User testing

Acting like the user

During the system testing a lot of acting like a user was already used to determine if the element programmed works as intended. At the end of the project we tested the system as a whole from the viewpoint of a user. For this we once again borrowed a lamp from the University's theatre department, loaded up a script and set up the lamps and other settings, plugged everything in and started the autocue to see how the play would go.

During this we found that our code did again run fine on Linux, but was not working on windows yet. Some differences in OS seemed to be needing to be solved again or were not taken into the program when the current build was being built. This was a good idea to test again at this stage, so that the program sent to the client for sure has all the bug fixes properly integrated.

Testing by user

The system was regularly shown to the client to make sure that no items were missing and that we were still on the right track. These meetings were usually planned when there was something to show and then held in the same week. Since our client was far away in Maastricht, we couldn't show the product in real life and since the client doesn't have a device we could build for at the time nor a DMX controller to test with we performed demonstrations as a substitute.

For these Demonstrations we ran the current product as it was and shared the laptop screen with the client, talking them through the process. All remarks during the demo were noted down and used to steer the product development after the meeting.

The final demonstration was on the 3rd of November, where we showed the final design of the product and the features implemented. This meeting was also used to verify the requirements in the next phase of this project. During this final demonstration the project group found some final little bugs to fix, which were fixed before the handover.

7 VALIDATION

For a good project ending we need to make sure that the product we built conforms to the requirements we set out. To do this we walk through the requirements ourselves to see if we see anything we missed, and we had a meeting with the client on the 3rd of November to go through the product and the requirements and see if he sees anything that is missing.

7.1 Requirements verification

The following table shows all of the requirements we laid out in the definition phase and printed in appendix B with their status as delivered in the final product. As to be seen, some elements are missing. These were either part of the “could haves” for this system and time ran out.

This list was also discussed with the client of this project in the final meeting. The client confirmed that the requirements that are marked as done are indeed part of the program as the client intended it to be there.

General	**Status**
The administrator should be able to load a .txt file formatted as a script into the program.	Done
The system should verify the input is of the right format	Done
The administrator should be able to start a play.	Done
The administrator should be able to edit settings for the play	Done
The system should end the play properly	Not Done
The administrator should be able to abort an ongoing play.	Done
The administrator should be able to enter different text formats to the system	Done
Backup script per role	
The system should be able to separate each character's lines and cues.	Done
The system should generate personal backup scripts for each role.	Done
The administrator should be able to vary the amount of lines that are visible before a line of the perform in the performers' backup scripts.	Done
The administrator should be able to export the backup scripts to a .pdf file.	Done
Autocue	
The performers should be able to read the script from everywhere on the stage.	Done
The performers should be able to advance the autocue.	Done
The performers should be able to see multiple upcoming lines on the autocue.	Done
The performers should be able to see previous line(s) on the autocue	Done
The administrator should be able to configure the font size	Done
Each performer should be able to advance the autocue.	Done
The performer should be able to quickly differentiate between characters lines	Not Done
Lighting	
The lighting system should adapt to the speed of the autocue.	Done
The lighting system should work with a set amount of lights.	Done
The lighting system should control the lights via the DMX protocol.	Done
The lighting system should create light on the stage.	Done
The stage should have some light at all times.	Done
The lighting system should work with a variable amount of lights.	Done
The lighting system should adapt to the script.	Done
The administrator should be able to enter the current lighting system configuration	Done
The lighting system should react to what is happening on stage	Done
The lighting system should work with a variable amount of varying peripherals.	Not Done

8 HANDOVER AND SUPPORT

The final steps in the project lifetime is handover and support. In this phase we make sure that the client receives our finished product packaged neatly and can start using it. For this phase we have to perform some tasks like building the application for running as an executable, such that the product is directly usable. We also made sure the code is complete and packaged such that anyone else with programming experience can continue working on it if needed. Finally, we make sure that any user can easily understand how to use the product with the help of a manual.

8.1 Manual

For using the product and modifying the code a manual has been written. In this manual is a clear description of what steps to take to use the product. It states what is needed to start running the application, how to input different settings, and how to run the autocue after that.

Next to this, the manual describes what is needed to run the application in a development setting. This includes requirements to run the application, and the commands needed to do so. It also contains some information about the structure of the code, so that it is easy to understand where you need to go looking for the thing you want to change.

9 DISCUSSION

In this chapter we will reflect on how the project went. That means that we will highlight some of the elements we saw could use improvement, discuss some feedback we got during the various presentations and reflect on the system of project reporting with phases.

9.1 Current limitations and possible extensions

Some elements of the project have had to be limited in their outcome to make sure that the project would be finished on time. A hard deadline needed to be set since this project is only part of a module, which runs the standard runtime of 10 weeks, and the result has to be delivered inside the module time to get a valid result at the end of the module.

This part of the report also describes some things another developer or our team could program into the system for it to be better, so some possible extensions. These are not the core requirements for this project but might be for the next iteration.

9.1.1 Lighting system

One clear limitation was the lighting setup as part of the lighting system. The lighting setup system had to have very generalized options for lamps and can only control lamps, because adding options takes implementing that option in the entire lighting control element of the program. Meaning that it needs new setup options, storage category and AI control for each new category. This means that we only have dimmers and RGB spotlights to convey the meaning of the play in the lights. But more kinds of lighting can be added in the future.

Another thing to mention is the lighting AI itself. Right now the system implements a very basic positive or negative detection system on account of the current line. This is just because that was the easiest to implement “AI” that could be found. A better system might be to detect actual emotion from the lines, with full training on different kinds of text and maybe some more granular control of the lights than we have right now.

This system could also be trained on Dutch text, so that it would be possible to perform a Dutch play with this application. This might be a bigger project though since we assumed English in all facets of the project.

A somewhat smaller fix available right now might be to average the lighting based on the results of the last couple of lines, or fading between the colors. But there was no time left to implement that since it was only thought of during the final meeting.

Spotlights are also currently not controlled, since we had no way of knowing which performer was where on stage, and or we had no time to make the lighting AI know what line was from which role so a special place per role was also not an option.

This can easily be added by either tracking the players somehow, or making sure the same character gets light on the same place by making the AI character aware. Making the AI character aware would require the AI to import this play's roles and determine a spot for them on stage. It might also be useful to increase the amount of blocks in the lamp setup so that more characters can be supported.

9.1.2 Autocue

The current autocue system has a simple to fix limitation with the fact that 2 simultaneous presses of a pointer will cause the autocue to skip forward 2 lines. In this project we decided that this was the expected behaviour, but a possible small extension can be made that blocks the input from the clickers a bit and makes sure that pressing it quickly will only result in advancing the autocue by 1 line, possibly unlocking after a certain amount of milliseconds. This could make for a possibly less chaotic play.

Another small extension possible to the autocue might be some differentiation between the lines of each performer. Currently the system only has 1 text color for all the lines on screen, making that the performer must read to see if their character has a line coming up, which is potentially disturbing to the play. A method might be to make the lines of each character have a different color. If this is too disturbing this can also be done with a little square in front of the lines in different colors for each character to differentiate.

A simpler version of this would be to just make 2 colors rotate between all the lines to easily see for on the autocue what lines are at least belonging to different roles.

9.2 Project setup and execution

This project had a somewhat different setup than we use normally in a project. Now, after the project is almost over, we have some remarks on the system we worked in and our implementation of it. As a reflection not about the results but the process.

To start with a short recap of the phases system. The idea was to perform every step in the design project as a phase, writing a report for each phase and then adding these together in the end to create a full report. This also meant that each phase was supposed to be a separate thing and thus worked on at separate times.

In the end the result of these phases being separate made us take a lot of time to work on the initialization and definition phase, and start quite late with our design and implementation. This caused us to have less time for working on the programming which we compensated by shortening the testing and validation phases and working harder in the final 2 weeks.

If we would have done initialization and definition faster and intertwined, we might have been able to do better testing or deliver a better product. But then also there might have been some more delay from not having the project properly thought of before we mindlessly started programming, which also generally causes delays further on in projects.

A RISK ANALYSIS

A.1 Working from Home

Risk: Medium
If another COVID outbreak takes place in The Netherlands, the government might mandate all education to take place online. This would implicate working from home for the whole project group.

Effects:
Technical difficulties during the meetings will hurt communication efficiency
Disconnect between team members will hurt productivity
Ease of procrastination and lack of oversight will hurt productivity

Measures:
Unfortunately, we don't have any influence on the government measures

Deal with effects:
Efficient planning leaves time to deal with arising issues
Clear expectations leave room for working independently and communicating effectively
Daily checkup moments make sure everyone can oversee the work that has been done by others

A.2 Illness of a team member

Risk: High
Members of the team being infected with COVID-19 or having other illnesses

Effects:
Team member has to stay at home, which could result in a less efficient work ethic
Team member cannot do the work they were assigned

Measures:
We don't have influence on the health of team members

Deal with effects:
Daily startup with all members (digitally) will help the group mentality
Redistributing the work among the other team members will make sure the absent team member can rest and heal without the project being delayed

A.3 No communication with the client

Risk: Low
Client is unreachable

Effects:
Where possible and thus the proper programming of the elements under testing will be checked
Transfer of information slower or nonexistent
Feedback from client on prototype(s)
Questions to client for clarification on requirements
Updating client on progress of the project

Measures:
Weekly scheduled meeting with client to ensure they are not busy at that time

Deal with effects:
Instead of showing the prototype, sending previews via email
Project proposal with proposed solution and setting MVP (minimal viable product) to ensure requirements are clear

A.4 Underestimating labour involved

Risk: Medium
Underestimating the effort required for finishing the product in time

Effects:
Falling behind schedule
Unfinished product at the final presentation

Measures:
Some team members are experienced in the used framework(s), so estimating the labour is more accurate
Team members are experienced in working in a project environment and are familiar with steps involved in the development process
Team members read into required hardware for setting up lighting plan since they haven't worked with this type of hardware yet

Deal with effects:
Put in more effort where necessary to ensure that deadlines are met

A.5 Remote development without hardware

Risk: Medium
No access to the hardware that will be used at the theatre festival

Effects:
No testing on location
Product might not work with the hardware

Measures:
Acquiring access to similar hardware for testing, in cooperation with the client
Developing product for generic hardware, such that small differences don't matter

Deal with effects:
Arrange hardware that works with product and let client use that

A.6 Other obligations of team members

Risk: Medium
One or more team members have obligations and/or appointments that interfere with the progress of the project

Effects:
Less time to work on the project
Unfinished parts of the project on the deadlines

Measures:
Close communication about work times and other obligations
Keeping track of work done by all team members and deadlines that were set

Deal with effects:
Compensate for hours lost during the week by working overtime on other days
Divide work over other team members

A.7 Communication barrier

Risk: Low
Miscommunication with client due to misunderstanding of jargon

Effects:
Wrong implementation of requirements and expectations

Measures:
Thorough involvement of the client in the progress
Project proposal includes detailed requirements and will be discussed in more detail with client before starting on the product

Deal with effects:
Work to fix the issues

B FUNCTIONAL REQUIREMENTS

General	
The administrator should be able to load a .txt file formatted as a script into the program	Must have
The system should verify the input is of the right format	Must have
The administrator should be able to start a play.	Must have
The administrator should be able to edit settings for the play	Must have
The system should end the play properly	Must have
The administrator should be able to abort an ongoing play.	Should have
The administrator should be able to enter different text formats to the system	Could have
Personal Back Up Script for Actors	
The system should be able to separate each character's lines and cues	Must have
The system should generate personal backup scripts for each actor	Must have
The administrator should be able to vary the amount of lines that are visible before a line of the actor in the actors' backup scripts	Must have
The administrator should be able to export the backup scripts to a .pdf file.	Should have
Autocue	
The actors should be able to read the script from everywhere on the stage	Must have
The actors should be able to advance the autocue	Must have
The actors should be able to see multiple upcoming lines on the autocue	Must have
The actors should be able to see previous line(s) on the autocue	Must have
The administrator should be able to configure the font size	Should have
Each actor should be able to advance the autocue	Should have
The actors should be able to quickly differentiate between characters lines	Could have
Stage Lighting	
The lighting system should adapt to the speed of the autocue	Must have
The lighting system should work with a set amount of lights	Must have
The lighting system should control the lights via the DMX protocol	Must have
The lighting system should create light on the stage	Must have
The stage should have some light at all times	Must have
The lighting system should work with a variable amount of lights	Should have
The lighting system should adapt to the script	Should have
The administrator should be able to enter the current lighting system configuration	Should have
The lighting system should react to what is happening on stage	Could have
The lighting system should work with a variable amount of varying peripherals	Could have

C QUALITY REQUIREMENTS

Autocue
The text font on the autocue should be readable from 10 meters away
Advancing the autocue should be straightforward for the performers.
The autocue should look consistent throughout the different plays.
The autocue should be comprehensible.
The system should advance the autocue within one second after a player has pressed the button.
Advancing the autocue should not interrupt the play.

System
The user interface should be intuitive.
The play should be ready to be performed within 1 minute after uploading the script.
The system should work on the device used at the festival.
The system should be able to handle scripts of up to 10000 lines.
The system should be able to work without failure for at least 5 hours.
Play does not have to be perfect (when compared to a ‘regular’ play).

Lights
The lights should not cause epilepsy.

Backup script
The backup scripts should be in easily readable format.

D DESIGN DIAGRAMS

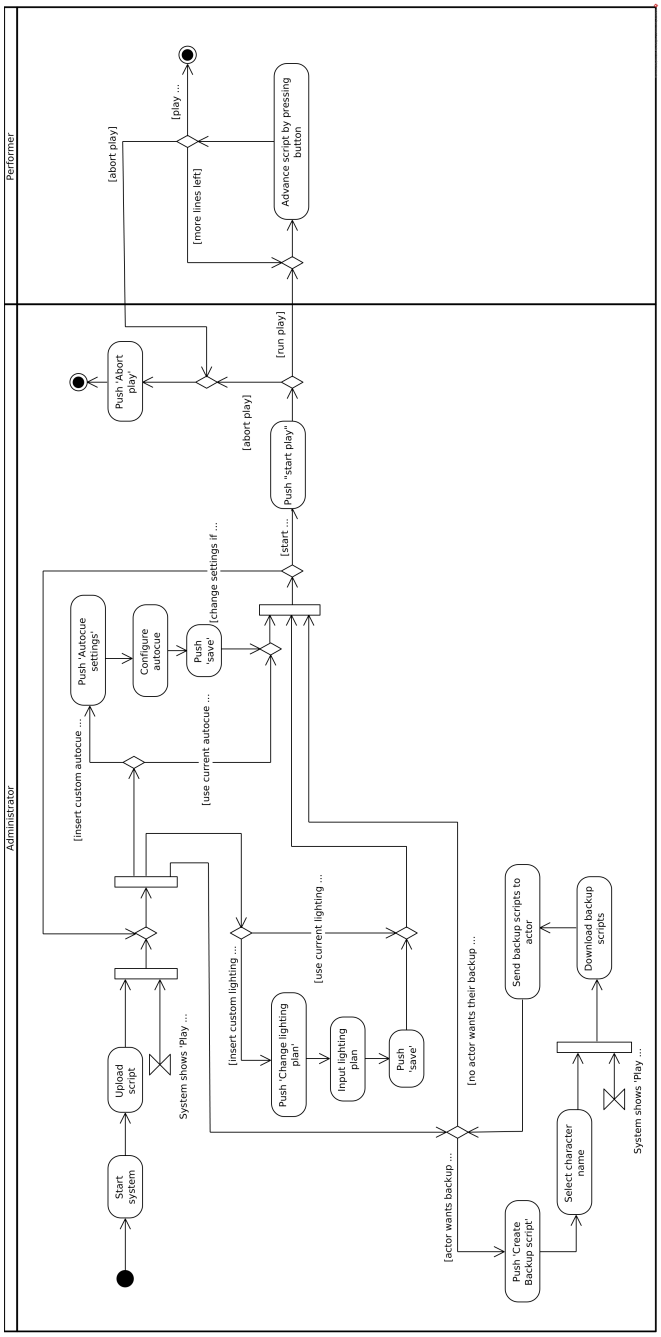


Figure D.1: Activity Diagram

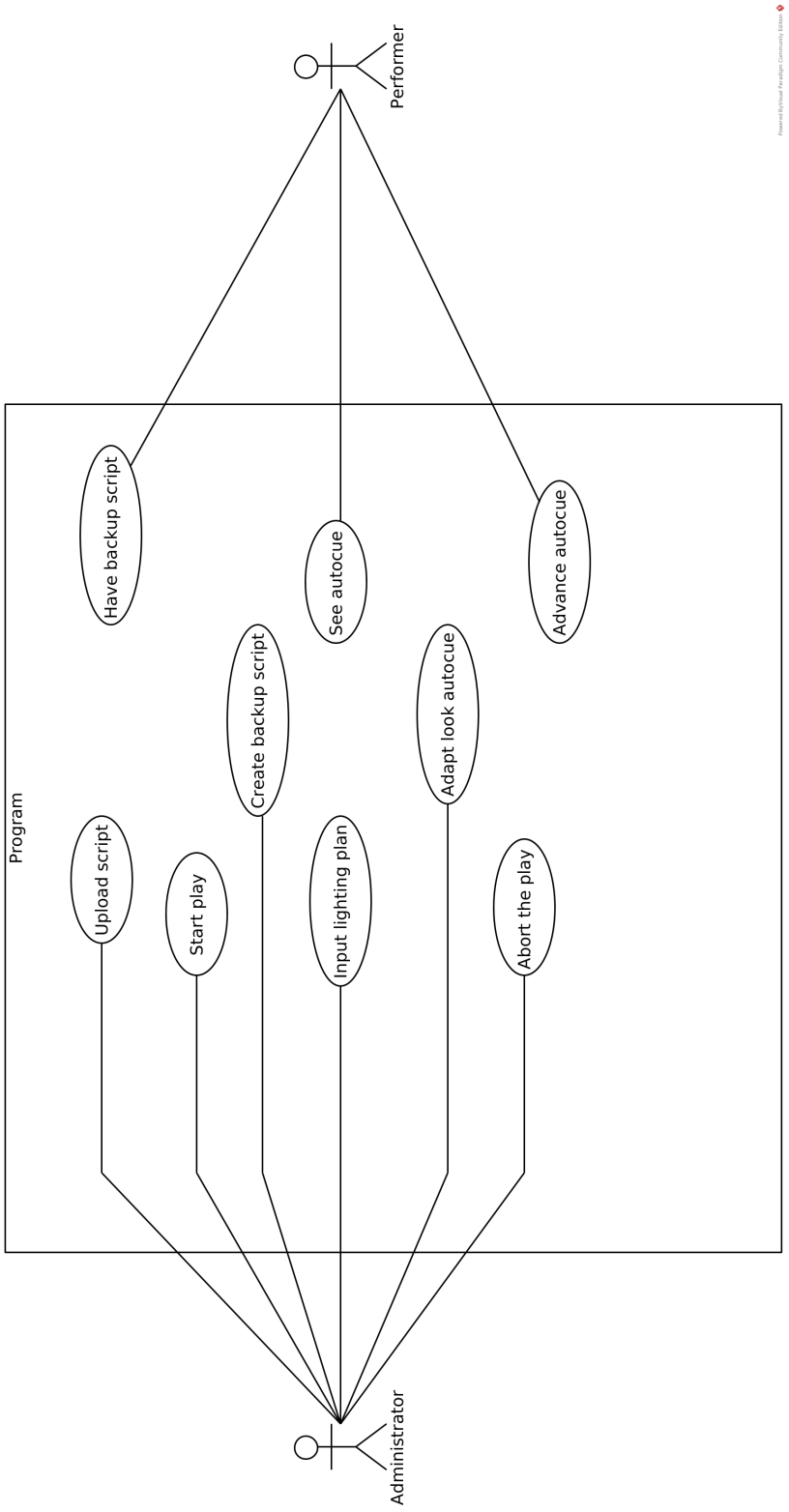


Figure D.2: Use Case Diagram

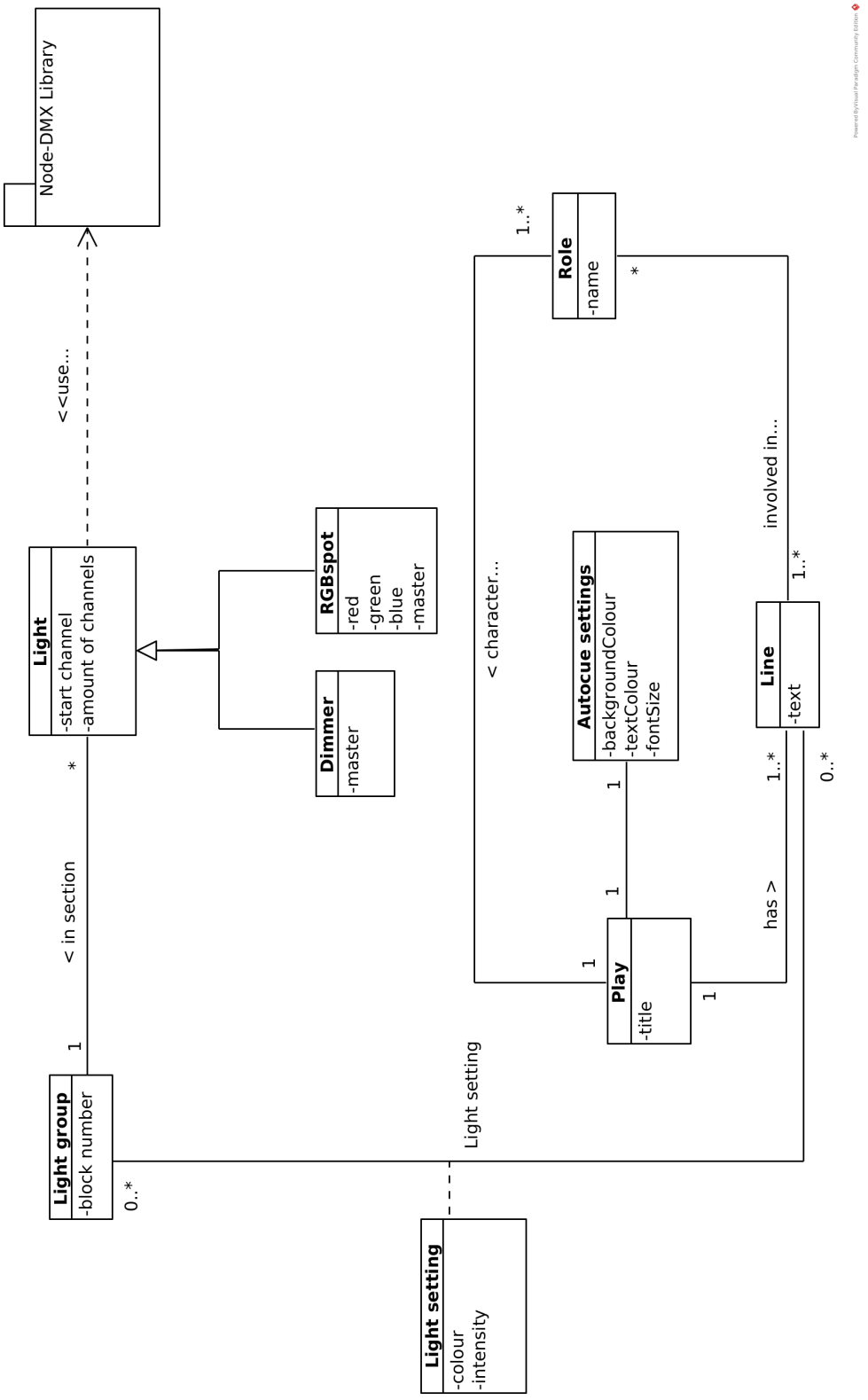


Figure D.3: Class Diagram

E APPLICATION DESIGN

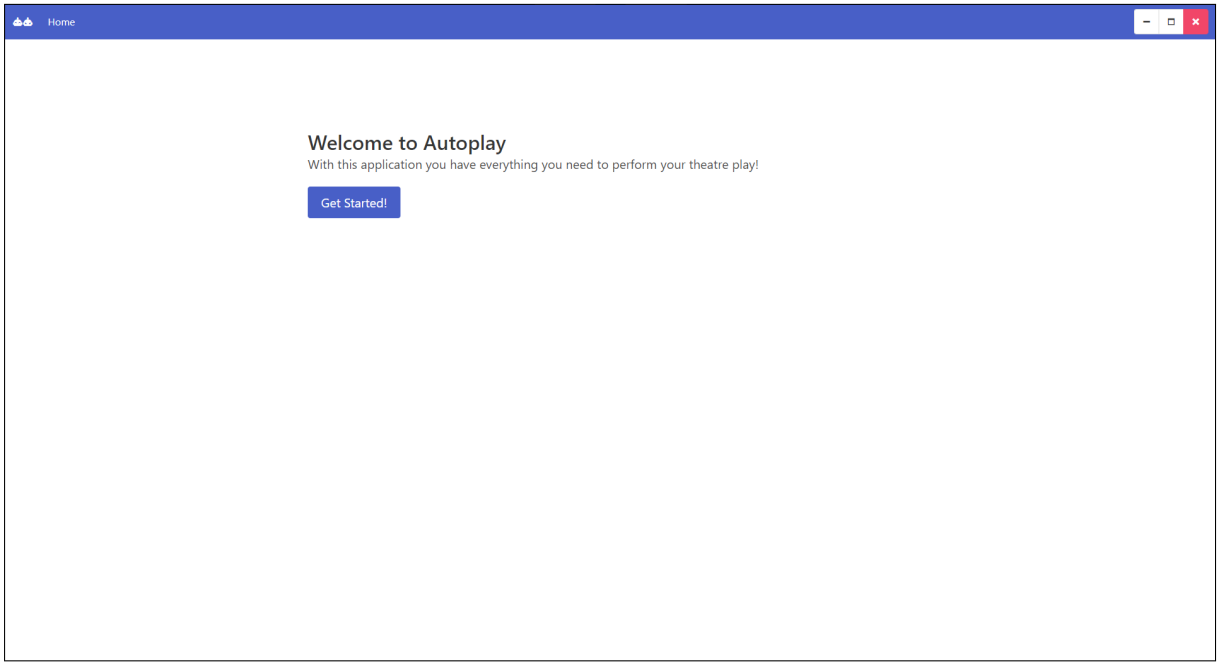


Figure E.1: AutoPlay Landing Page

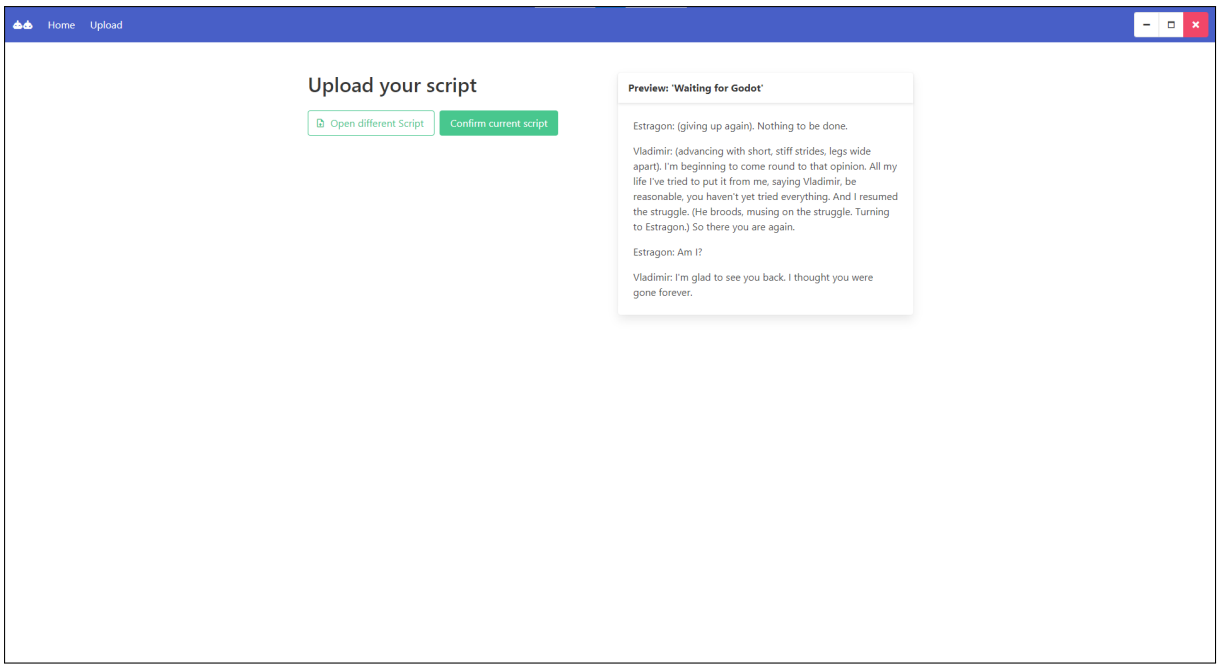


Figure E.2: AutoPlay Script Upload Page

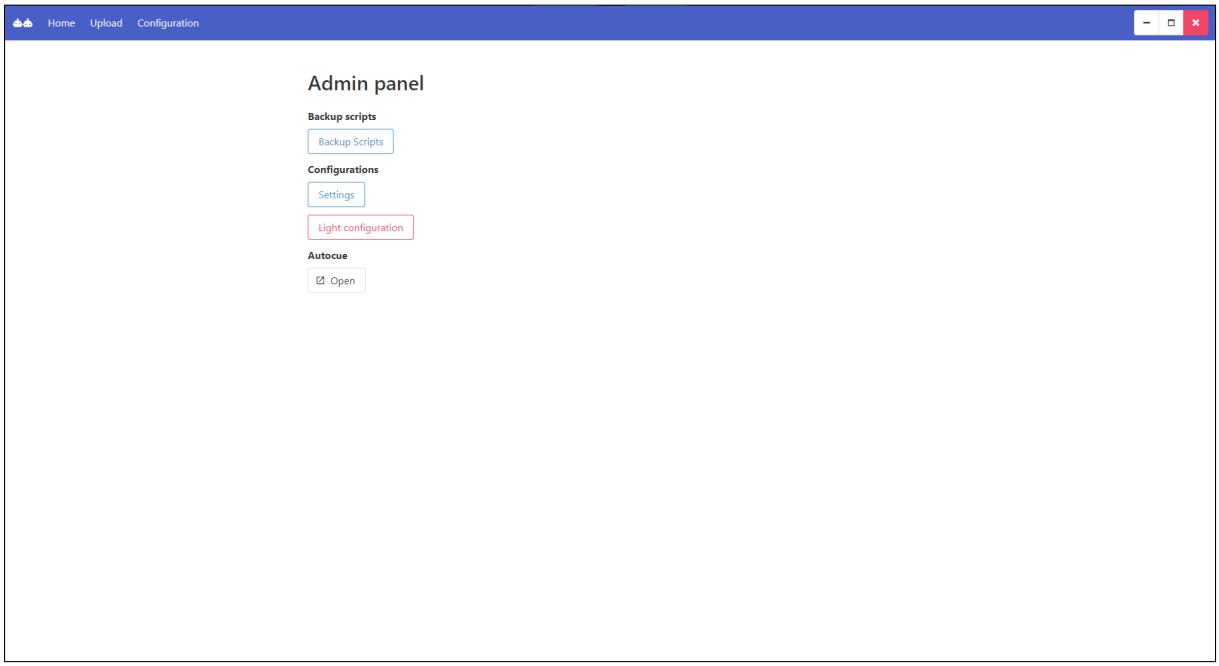


Figure E.3: AutoPlay Admin Page

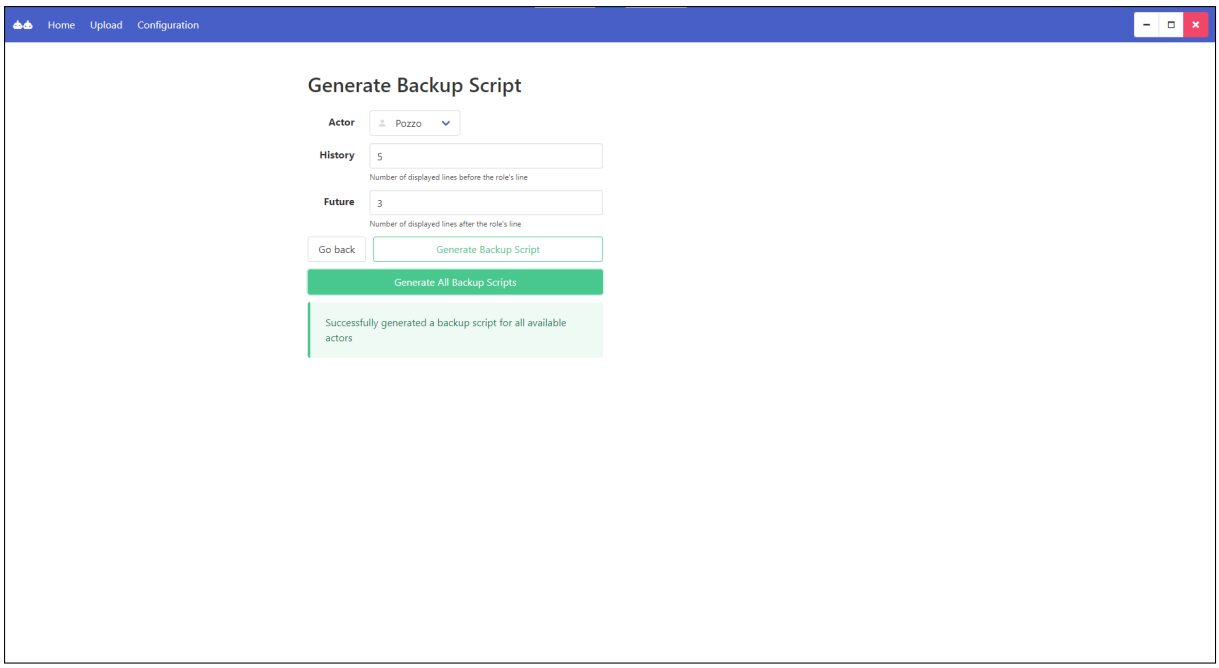


Figure E.4: AutoPlay Backup Script Page

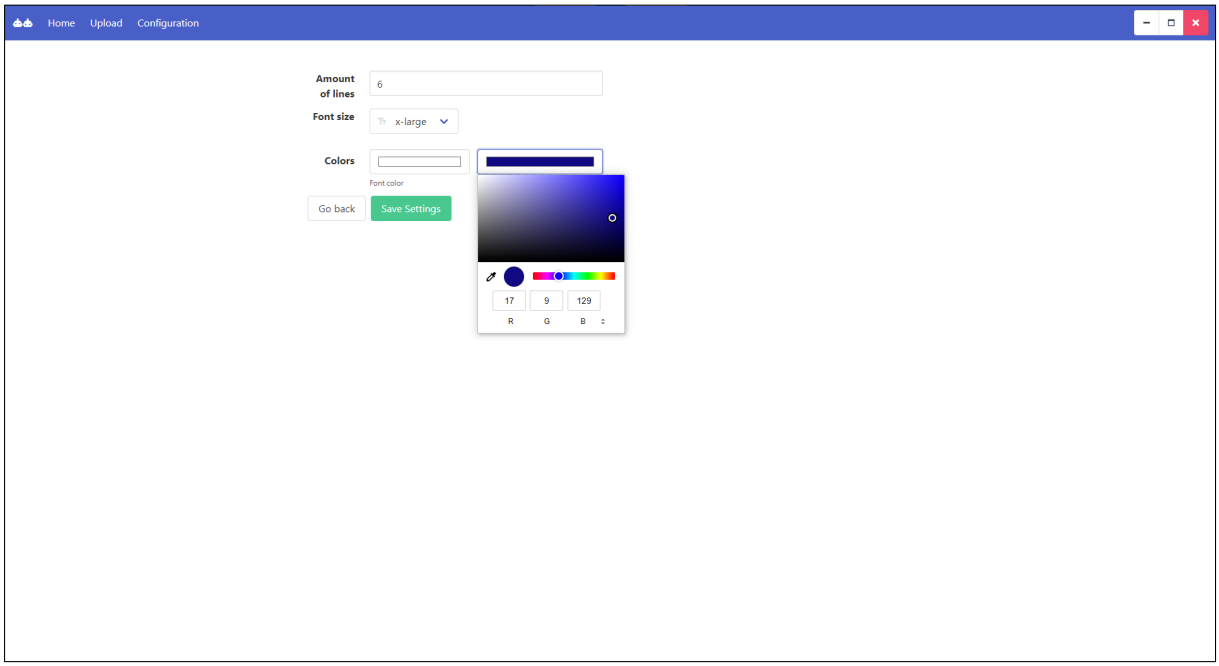


Figure E.5: AutoPlay Autocue Settings Page

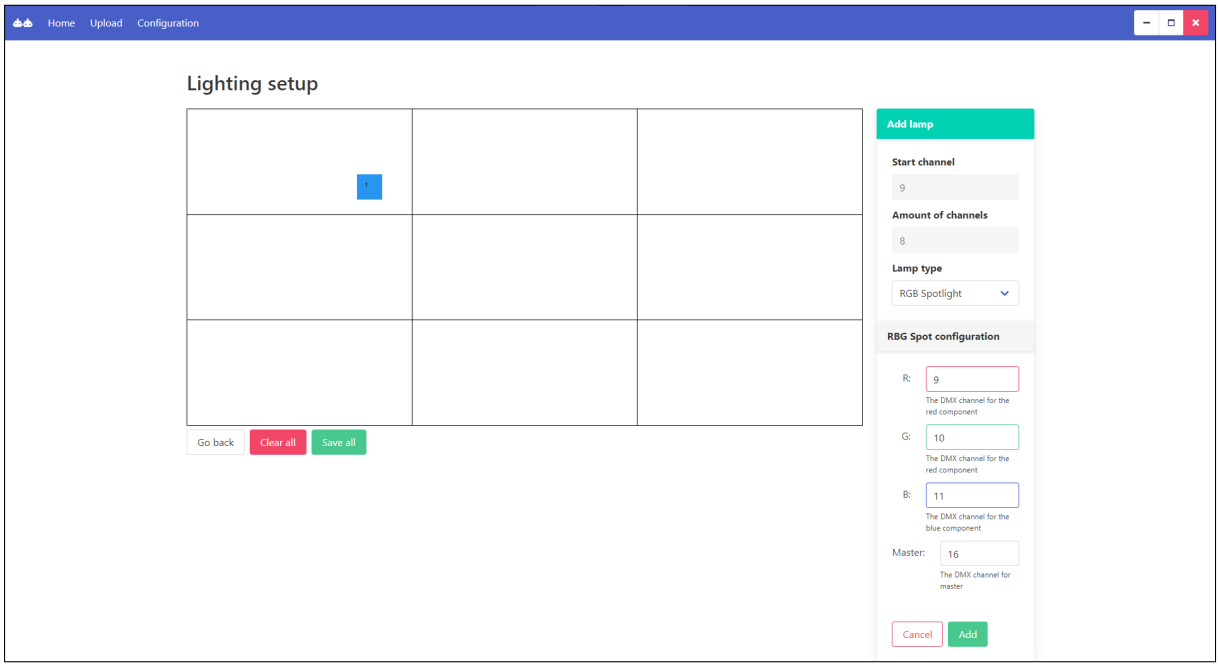


Figure E.6: AutoPlay Lighting Configuration Page

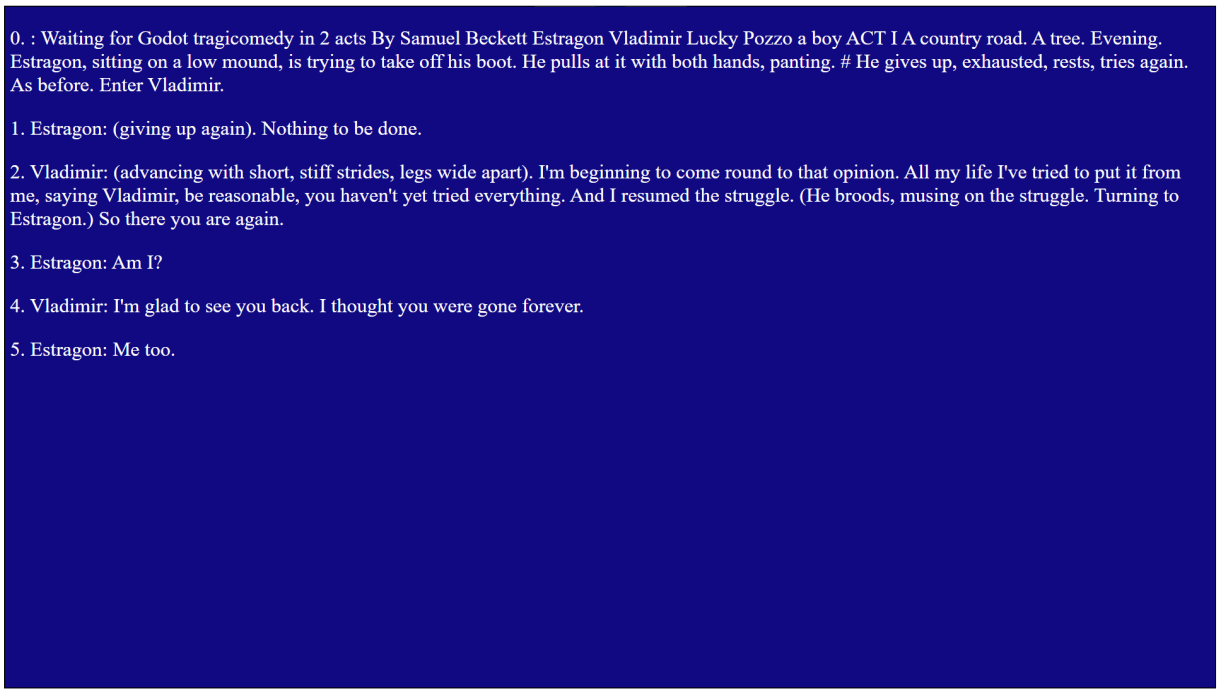


Figure E.7: AutoPlay Autocue Example