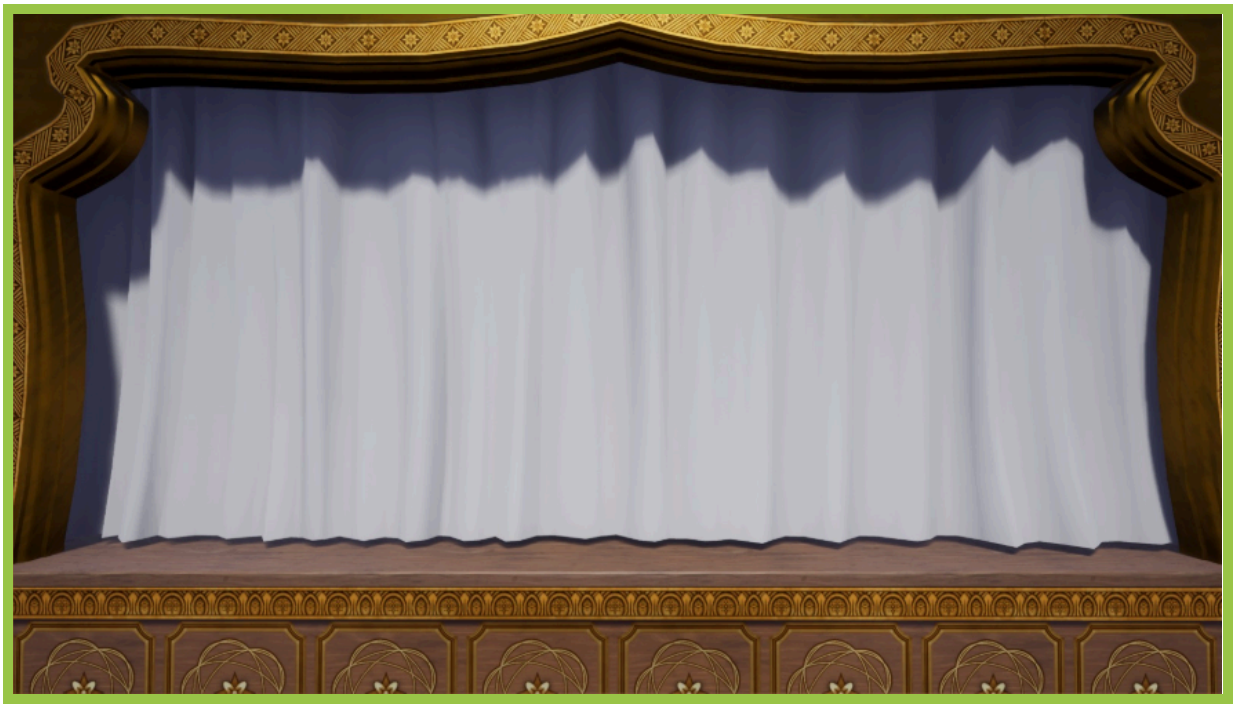
Show Stopper



- Show Stopper Risk Assessment -

Andrew Rimpici - Jacob Biederman

- CCC-410-01 -

TABLE OF CONTENTS

DIFFICULTY SCALING SYSTEM

Document Scaling System

Scaling Overview

- This document uses a 1 through 5 difficulty scaling system where 1 represents very easy, and 5 represents extremely difficult.

Difficulty Representation

- The difficulty will be represented using the Action Cactus Studio logo shown below:

	**Represents 1 out of 5** Very Easy - No trouble to implement
	**Represents 2 out of 5** Easy - Little trouble to implement, but may take more time than usual
	**Represents 3 out of 5** Medium - More thought than usually needed for planning and implementation
	**Represents 4 out of 5** Moderately Difficult - A lot of planning and time needed for planning and implementation
	**Represents 5 out of 5** Very Hard - Very difficult to plan and would take too long to implement before the deadline

TECHNICAL ROOTS

Unreal Engine 4

Unreal is a well established game engine originally designed for first person shooters. It supports our core technical needs in networking, multiplayer, physics, and controllers. It also features a clear programmer vs designer workflow in the separation of C++ and Blueprints. It is a risk because of our lack of experience, but there is a multitude of online support in the form of tutorials and documentation. Our game is rooted in the 3D Sidescroller template that comes default with Unreal.

Version Control - Git

Git is one of the major players in version control and our only possible choice is between it and Subversion. Git's branching model does not work well with Unreal blueprints which are designed to work with Perforce. However, out of the available options it is the one that we have the most experience with.

Coding Standards

Our blueprint coding standards are rooted in both Object Oriented and Functional Programming. Blueprints have a high tendency to turn into Spaghetti code which we will counter with the standard known protocols: moving behaviour into functions, using local variables to remove long connecting lines and using good naming to denote what functions do.

Twitch

Twitch integration is more recent with our experience rooted in Twitch Plays and Vermintide II both of which allow viewers to influence the game through keywords in chat. Our technical roots are in the MIT licensed TwitchPlay plugin by TheDiG3.

THE DELIVERY PLATFORM

Main Delivery Platforms

Computer/Console Platform

- *Show Stopper* is a party game designed for the computer and using console controllers as input. Currently, our main delivery platform is personal computers as they are easy to develop for and allow for a standard workflow. Consoles are ideally the next delivery platform to extend support to when the time arises.
- One downside to targeting personal computers first, while also requiring console controllers, is that all players playing on personal computers may not have console controllers readily available to them.

THE DEVELOPMENT ENVIRONMENT

Programming Environment with Unreal

Using The Unreal Game Engine

- The game will be developed in the Unreal Game Engine. Unreal comes with a lot of built-in features that are right at the hands of the developers.
- About half the team is comfortable using Unreal while the other half will have to learn it, which is a huge technical risk.
- Unreal is also extremely useful because it allows for cross-platform development.
    - This comes in handy as the developers can simply export the game to a console version of the game with little overhead—*saving time and money*.
- Since the programmers are not as familiar with Unreal, it will be a big challenge to develop efficiently and produce high quality content before the deadline. In our case, we think the advantage of learning a new development environment outweighs the risks.

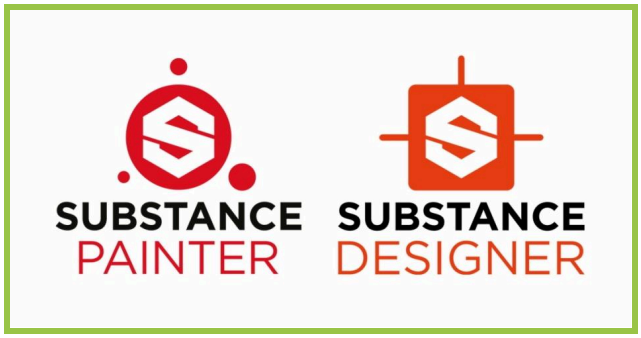
THE DEVELOPMENT ENVIRONMENT

(Continued)

Resource Environment

Substance Designer/Substance Painter

- Michelle our artist uses Substance Painter and Substance Designer to develop for most of her texture needs.



- She uses Substance Painter primarily to create unique prop texturing as she is most familiar with this program for this discipline and can create assets efficiently using this program.
- In addition to Substance Painter, she also uses Substance Designer to create procedural textures which will be helpful when creating a variety of stage props and character clothing options.

THE DEVELOPMENT ENVIRONMENT

(Continued)

Resource Environment

Maya

- Our artist, Michelle, uses Maya as her main program of choice to create 3D models for the game. Michelle uses Maya because she can create high quality assets in a more efficient amount of time than she otherwise would if she were to use a different program.

THE DEVELOPMENT ENVIRONMENT

(Continued)

Version Control

Git

- The choice of version control for this project is Git. In Pineapple, there is a git repository linked where all of the team members can monitor and update the game files. The use of Git will help the team make sure all of the game files are up-to-date for the required deadlines.

GAME MECHANICS AND SYSTEMS

Dieing on the Opponent's Sword

- The core gameplay of Showstopper is the ability to die on the other players' sword. The gameplay revolves around this idea and creates a unique experience for players used to more traditional fighting styles in other games.

Audience Favor System

- The audience plays a major role in the gameplay and can influence how the action on the stage pans out. In the future, we plan to integrate twitch livestreams into the audience system to allow for real life players to alter the game from the point of view of the crowd.

Sound and Vibration System

- Sound and haptic feedback are very important to keep the players engaged in what they're viewing on screen. Using haptics for the player controllers we can help keep this immersion on top of implementing key sounds during important gameplay moments.

Game Modes

- Unreal uses a technology called Game Modes. Jacob and I are going to have to research more into these and see how we can use these to their full potential.

Fighting/Posing Mechanics

- The game relies heavily on movement and combat feeling good. If these systems don't feel good, then the whole game will fall flat. Another mechanic that is integral to the core experience of the game is posing. Since players are actors on a stage in a play, they should be able to pose! These poses can rally the crowd or give them other buffs depending on the current play they are reenacting.

TECHNICAL RISKS

Adding more than two players

- Adding more than two players to a local game should not be a challenge. There are a series of changes we will need to make to our code to account for the additional controllers, but beyond figuring out controller IDs, this should be simple.
- Fix death architecture so that game doesn't end when first player dies.
- Player death and score is set up for two players.

Multiplayer / networked

- Multiplayer networking is the biggest unknown. We believe that by integrating our changes in with the Unreal character controller networking should be straightforward. While we're not making an fps, Stage Fight has fairly simple mechanics that should be easy to network.
- There are many tutorials we can follow, including: https://www.youtube.com/playlist?list=PLwmGmCVti_dDl8CMa_91FdwgZ0W010d5v which goes through setting up a basic networked game.
- This fits Andrew's interest in networking.

Twitch Chat Integration

- Integrating commands from twitch chat into the game seems like it would be a challenge. However, there are both paid and open source plugins that already allow this, and the connection technology is fairly simple.
    - MIT licensed plugin https://github.com/TheDiG3/TwitchPlay
    - Supports version 4.21, Last updated 1 year ago.
    - Connects with Unreal FSocket to twitch IRC.
    - Code is well commented, and there is adequate documentation.
    - This fits Andrew's interest in networking and is a great intro to networking.

Unreal

- Using unreal engine is our biggest technical risk. Neither Andrew nor Jacob have previous experience using it. Beyond requiring quick learning of a new system, it also has other technical risks.
    - Version control is difficult with both Unreal and Unity. However Unreal cannot serialize blueprints which means that it is hard to merge them. It is

designed for Perforce with file locking, which we can manage with good programmer communication.

- Programmer workflow will be another learning challenge and may be a frustration as we figure out C++ vs Blueprints.

Audience System

- Creating an audience system that can read and engage with the players performance in a meaningful way is a huge challenge. It is of course a design challenge and will require testing and iteration. It is also challenging from the systems perspective. Tracking the player's sword position and movements and using that data to determine the engagement level in the simulated audience will be difficult.
- This fits Jacob's interest in AI and simulated Audiences.

THE ART PIPELINE

Creating and Loading the Art

Creating the Art

- The artists create the art/models in the tool of their choice. The raw art files can be saved in the *Art* folder in the repository.

Loading the Art into Unreal

- After the necessary art assets are created, the artist can then add the final versions to the Unreal Assets folder for use by the Designers and Programmers.

Uploading Art to the Repo

- Once all of the assets are in the artist's local repository folder, they can push all of their changes to the Pineapple remote repository.

THE DESIGN PIPELINE

Unreal Engine

Blueprint Interface

- Unreal has blueprints which are extremely helpful to designers as they give them a tool to implement many features quickly and with little programming overhead. Our designer, Tim, in addition to our programmer Jacob, will mainly be using blueprints to get all of the major features in quickly.
- My main goal is to transfer some of the existing core features like movement from the blueprint code into C++ and then provide functions for Tim and Jacob to use in blueprint. I will also be creating new custom nodes as well as implementing core subsystems of the game in the C++ backend to make frontend blueprint development smoother.