TyrantVC, Version Control System for Maya

Group members: Carson Wilk, Volodymyr Loyko, Annie Gesellchen, Ben Celsi

Problem

Maya is a modeling and animation software created by Autodesk that is used widely amongst animation and gaming studios. Most of these studios have scripting teams that create specialized plugins and tools. In order to write these scripts, they work in Maya's built-in script editor [1]. While developers can work in an IDE, scripts can only be run and tested within Maya, so for the sake of efficiency and familiarity, developers generally only work in the script editor within Maya. However, this script editor is lacking in a lot of basic functionality that a programmer would expect from a typical IDE or code editor. Part of the reason for this is that many of the people who write scripts for Maya come from an animation background, not a programming background, so any add-ons or plugins to the script editor must be simple, easy to use, and intuitive.

One major feature that the Maya script editor lacks is a version control system. As a result, developers who want to use a version control system for their scripts would have to switch to an outside platform such as GitHub Desktop. Most developers don't do this for the same reasons that they don't use an IDE - switching is inefficient, and the platform is unfamiliar. TyrantVC is a plugin that will be added to Maya that will integrate directly with the script editor and give access to version control to script developers by providing an interface to Git.

Maya Background

Each Maya project is a single file. When writing a script for Maya, typically the user will open a Maya project and begin writing the script in Maya's script editor (fig. 1). These scripts are usually saved in a centralized folder so that if the user opens up a different Maya project, the same script could be opened and run in that project as well. There are two languages for scripting: Python and MEL [3]. There is also a wrapper library for Python called PyMEL, which allows developers to run MEL commands from within Python. Both scripts and plugins (including our TyrantVC plugin) are written using Python, PyMEL, and MEL. A plugin can consist of multiple script files, which for the purposes of TyrantVC would be considered a repository.

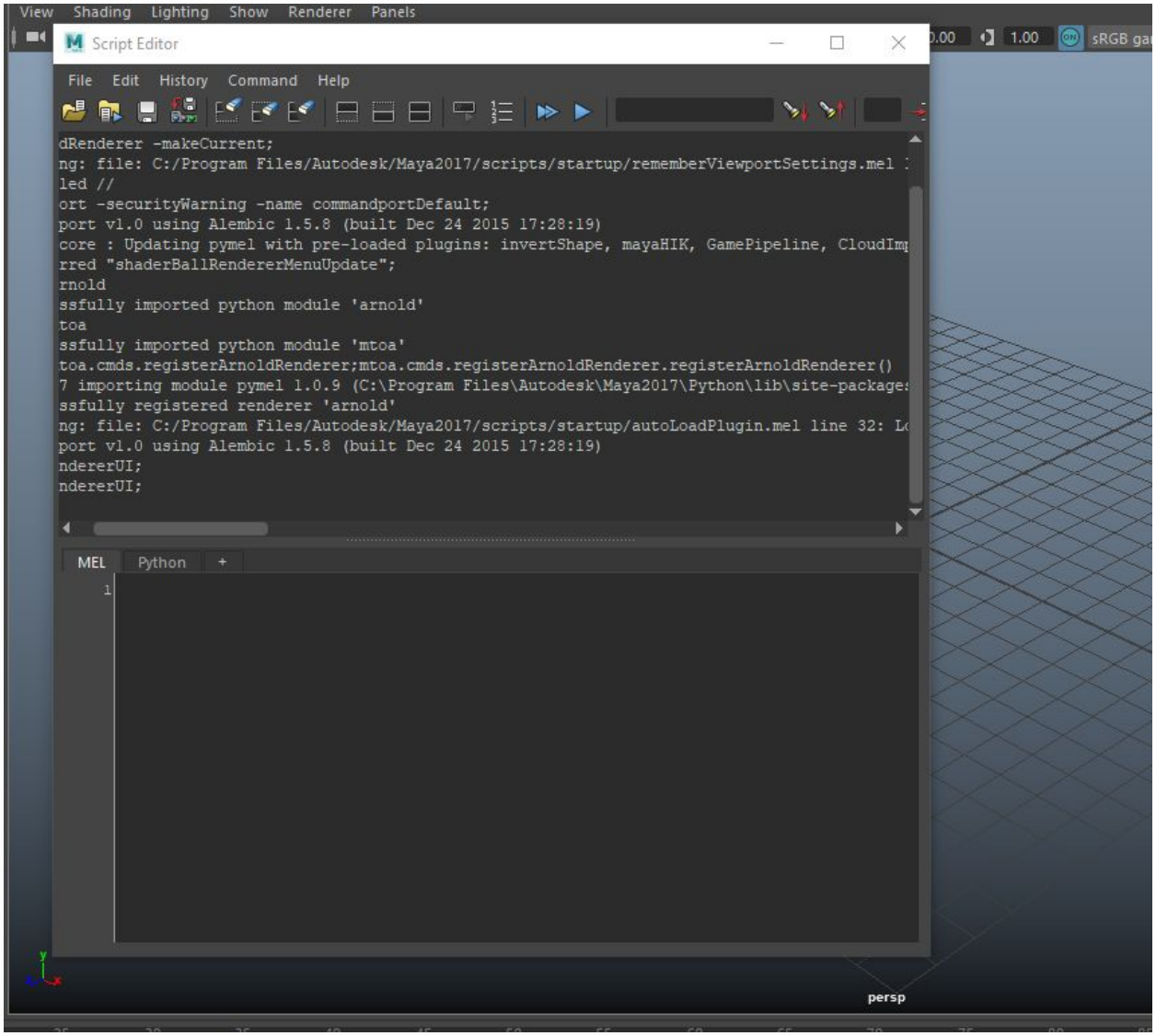


Fig. 1, a screenshot of Maya's script editor. The top panel is the output, and the bottom panel is where scripts are written.

An example of a typical Maya project that a script developer would be working with is a "rig", which is a skeleton for a character that will be animated. Scripts can be anywhere from a few short lines to multiple thousand-line files, and typically are written to help improve the efficiency or usability of some part of the animation pipeline. A possible script that could be written is one that would take a skeleton that has been created in Maya and automatically generate blocks around each bone, so that an artist can begin sculpting the 3D model of the character (fig. 2).

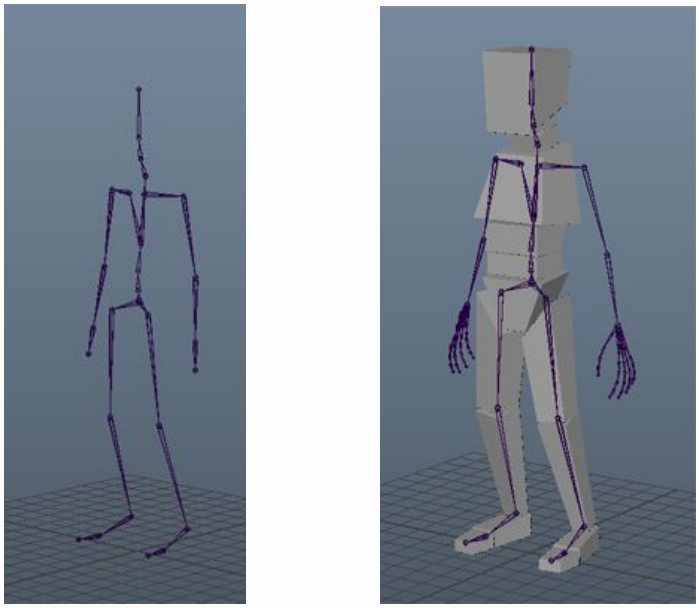


Fig. 2, visual aid of Maya scripting example -- applying blocks to skeleton.

Because all Maya scripts affect some part of Maya, they must be run and tested within Maya. This is why developers typically write scripts within Maya's script editor -- it makes the workflow and testing process much simpler than writing in an external IDE.

Motivation

After talking to a script developer who has worked extensively with the Maya script editor, we found that one of their biggest problems is that it can be very inconvenient and inefficient to save out versions of the scripts they are working on and to keep track of changes. This is because any scripts must be run in Maya in order to test them, and if they fail and cause Maya to crash, the developer could lose any work they had in the script editor. It can also be cumbersome to save iterations of a file without any version control system, and working on scripts from multiple computers can be difficult or create merge conflicts with no way to resolve them. For now, we are building TyrantVC as a single-user version control system because our target audience doesn't generally work in teams, but it could be later expanded to multiple users. Finally, many Maya script developers don't know how to use Git outside of Maya and would not be likely to try to learn a new system. Creating a straightforward way to access a version control system from within Maya, which can integrate directly into the existing workflows of script developers, would help alleviate a lot of these problems.

Currently, there are plugins that allow for auto-completion, better access to documentation, auto saving, and many more features, but no plugin for Maya implements version control for the script editor. One such plugin that we looked at is Charcoal Editor [2], which has many useful IDE-style features but does not provide version control and also costs $80. If developers want to

have version control, they must leave the Maya environment and switch to another platform such as GitHub Desktop or use git on the command line. This results in a messy and inefficient workflow. Additionally, because our target audience is coming from an animation background rather than a programming background, most users won't have prior experience with version control systems, and more specifically, git. This means that a system such as GitHub Desktop, which contains git functionality that our target audience won't need such as branching and merging, may be confusing to users. Furthermore, if a user were to try to interface with git on the command line, they would have to learn bash and git commands, which most users won't want to take the time to do.

Our plugin will focus on version control for script files rather than project files. This is because the scripts that are written for Maya are not specific to the project that the developer is working in, meaning that a script that a developer writes could be used with any Maya project. Because of this, using version control on the Maya project file (rather than on the scripts themselves) would be ineffective and would not solve the core problem. Major studios likely have created their own plugins to improve the Maya script editor, possibly including version control, but none of these are made public.

Approach

We have created a plugin for Maya that adds an additional UI panel to the Maya Environment. This UI panel allows users to see which repository they are working in, which files have been modified, and allow for standard Git functionality of committing files and viewing past versions of a committed file. It uses Git locally on the user's machine, not GitHub.

User Interface

Fig. 3, basic Maya scripting workspace. The highlighted panel is the TyrantVC plugin.

The user interacts with the plugin through a panel that defaults to being placed on the left side of their Maya workspace. Maya users can rearrange the panels however they see fit, but the image above provides an example of what the user's workspace could look like. The plugin has two tabs: one for viewing the files that are currently being worked on and one for viewing the history of commits for the current repository.

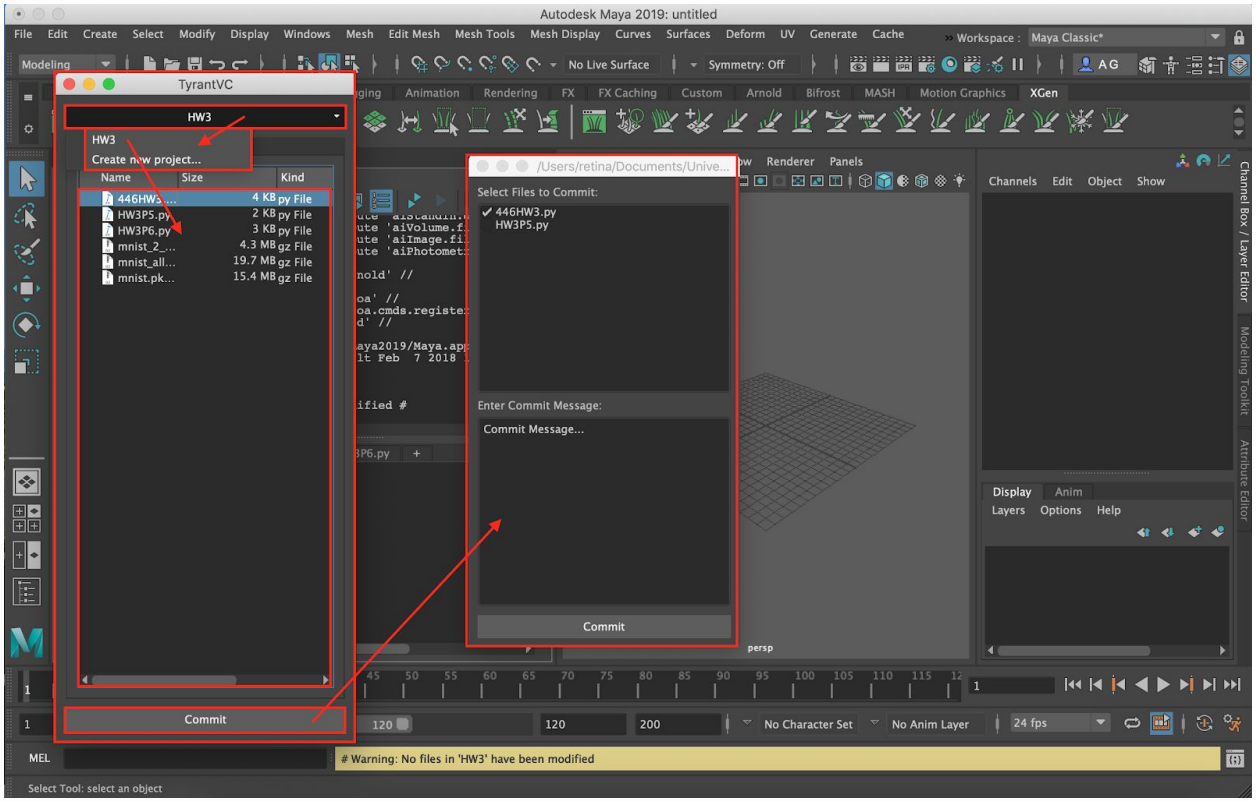


Fig 4, TyrantVC Files tab UI mockup diagram.

In the files tab, the user can see the file structure of their current repository. When the Commit button is clicked, a window pops up allowing users to select which files to commit and provide a commit message. Users can also click on the repo dropdown at the top of the plugin to select a different repo, or add a new repo. Selecting or adding a repo populates the file tree. In the file structure, double clicking on a file will open that file in the script editor. Selecting a file by clicking on it once allows users to see more information about that file as well as view the commit history for that specific file. Clicking the history button will replace the file structure view with a list of commits for that file. Here, users can view each version of the file and click on any one of them to open a pop-up window displaying that version. Users cannot edit the earlier version; it is read-only.

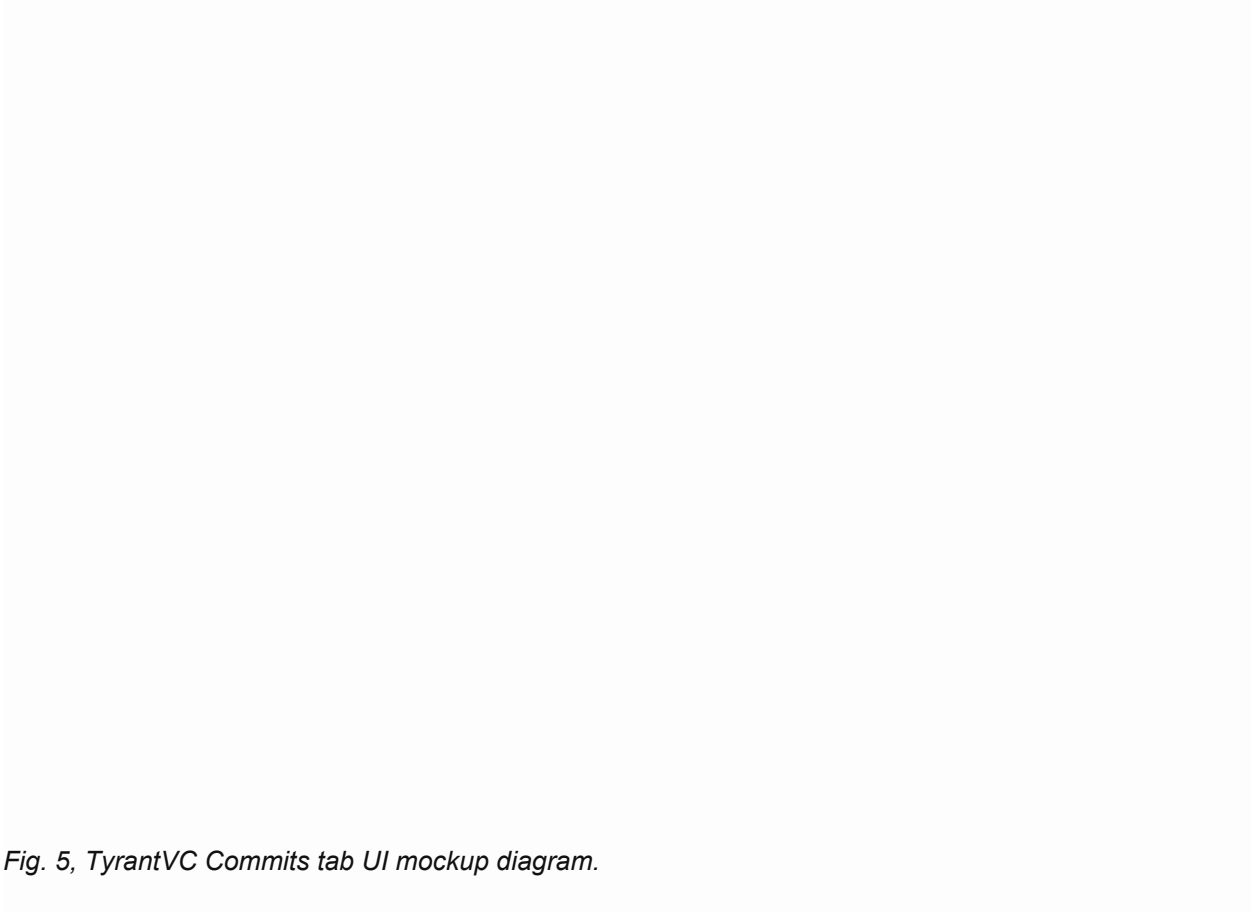


Fig. 5, TyrantVC Commits tab UI mockup diagram.

In the commit history tab, users can see each commit. This provides essentially the same information as in the files tab, but organized by commit rather than by file. Here, users can click each commit to view the files changed, as well as view each specific version of the file.

Implementation Details

The plugin is written in Python and tested using Maya and its script editor. The plugin consists of three major elements: the UI, connection to git, and the connection between the UI code and the git code. The UI component is written in using PyQT for Maya and will resemble GitHub for Desktop functionality [4]. GitHub Desktop should not be ported directly into Maya, however, because it contains much more functionality than is necessary for our intended audience. It has been simplified down to stating which repository you are working in, which files have been changed, and buttons for pulling, committing and cloning/making a new repo. The component that allows the connection to git is written in Python. This component needed to be tested on various operating systems since access to bash shell in Maya varies between UNIX based systems and Windows [5]. Finally the component that connects the two previous pieces

together is written in PyMEL, Maya's python wrapper for MEL. The purpose of this component is to connect the Python code and PyQT code into a working plugin for Maya.

Architecture:

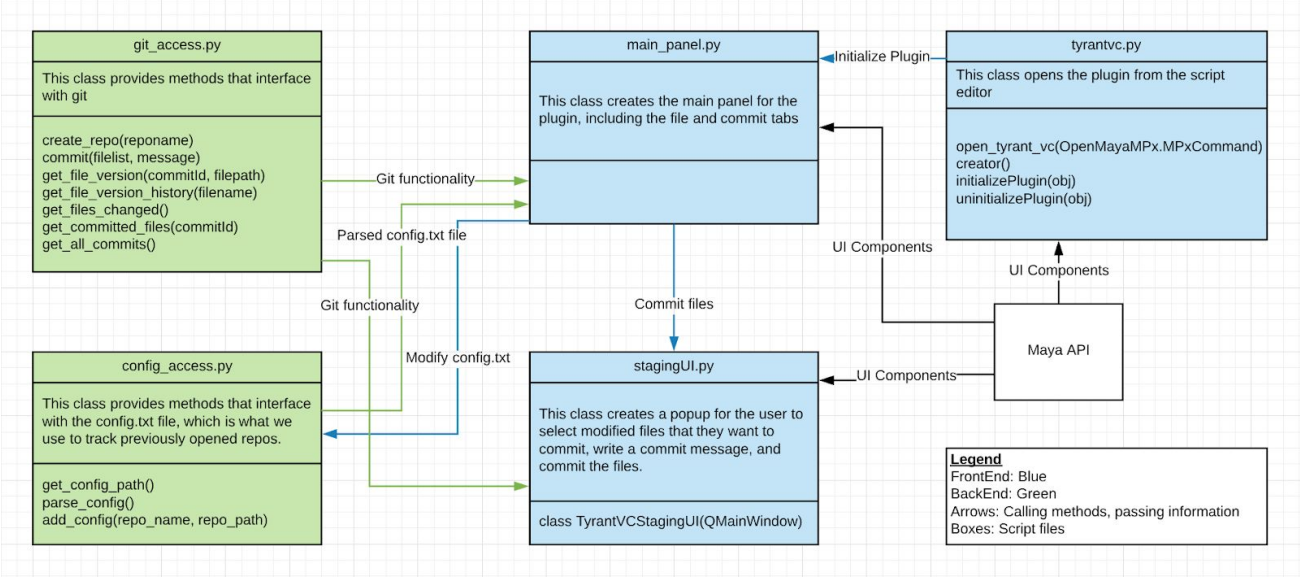


Fig. 6, TyrantVC code architecture diagram

We split our architecture into a number of front-end and back-end components. Front-end components are in blue and include the following classes:

- **tyrantvc.py:** This class is what opens our plugin. It is run from the script editor, and kickstarts main_panel.py, which is what controls / mediates between all other files.
- **main_panel.py:** This class creates the main panel for the plugin. It is called by tyrantvc.py. It includes the file and commit tabs, a repo selector, and a commit button. It calls stagingUI.py whenever the user click the commit button. It calls config_access.py to load the repo selector, or when a new repo is created. It calls git_access.py for all git functionality, for example getting all previous commits.
- **stagingUI.py:** This class creates a popup for the user to create a commit. In the popup, a user can select which modified files they wish to commit, and then add a commit message. To commit the files, a commit button is created at the bottom of the panel. This class calls git_access.py to get modified files and commit files.

Back-end components are in green and include the following classes:

- **git_access.py:** this class provides methods that interface with git. It is called by main_panel.py and stagingUI.py.
    - **create_repo(reponame):** creates a repo, or loads it if it was already created
    - **commit(filelist, message):** commits a list of files with a commit message
    - **get_file_version(commitId, filepath):** gets a version of a file at a given commit
    - **get_file_version_history(filename):** gets commits associated with a given file
    - **get_files_changed():** returns a list of all changed files
    - **get_committed_files(commitId):** gets all files associated with a given commit
    - **get_all_commits():** returns a list of all commits along with their dates, messages, and the files associated with them.
- **config_access.py:** this class provides methods to interface with the config.txt file, which is what we use to track previously opened/loaded repos.
    - **get_config_path():** this returns the absolute path of config.txt
    - **parse_config():** this returns a list of tuples from config.txt, with the first element of the tuple being the repo name, and the second element being the repo path
    - **add_config(repo_name, repo_path):** this adds a repo to config.txt

Teams

We have divided our project into 2 teams: Frontend (Carson and Ben) and Backend (Vlad and Annie). Frontend is responsible for building the UI using PyQT, and Backend is responsible for implementing an API of git commands, using PyMEL and for implementing the components that connect the front-end UI components and the git command API using Python.

Evaluation

Our goal is to provide version control to Maya script developers in a seamless way and improve the efficiency of their development experience. There are two parts that we will want to evaluate: the correctness and the functionality of our plugin. The correctness is tested through comparison to a predefined specification file which defines all of our plugin's functionality. An example of this is the drawn diagrams of what our plugin should look like, and the descriptions in this report of our plugin's functionality. Some of the evaluation, such as testing that our backend files can interface with git correctly, is done with automatic testing. Others, such as running through a user story where a user commits a file, are performed by hand. For example, since all MEL commands produce some output, we can compare expected output with actual output. However, since some of the tasks are more qualitative (e.g., did the user click the right buttons to accomplish the task), those would need to be verified through user studies. To test sufficiency of the functionality we will perform usability with a small user base. We have access

to the Animation Capstone students and staff who we will ask to perform a series of tasks using our plugin.

Examples of tasks that we will ask the users to perform:

- Commit a changed file to the repo
- Given a pre-existing script or set of scripts, create a new repo and add the script files to it.
- Find a specific previous version of a file

We will log all of their actions and see whether they are able to perform these tasks successfully or run into problem with our code. Additionally, we will ask about sufficient functionality, since we are providing a bare-bones version of git to increase the simplicity of our plugin. We have already discussed this user study with them and they are willing to help us out and are excited to see what we come up with.

Initial Evaluation

Our initial evaluation provided valuable feedback on how our product could improve. We tested 3 users on five tasks: installing our plugin, creating a project, editing a file, committing one file, and committing a different file.

Users were given the following instructions to follow:

1. Visit TyrantVC page (https://github.com/cagesellchen/TyrantVC) and install the plugin following the instructions in the README.
2. Create a new project using the plugin from the test folder located on the Desktop.
3. Modify files in ScriptEditor: *1.py* to `print(11)`, *2.py* to `print(22)`.
4. Commit 1.py with a message “11”.
5. Commit 2.py with a message “22”.

We recorded whether or not they were able to accomplish the tasks without guidance. Because our evaluation is a user study, it can't be scripted. Therefore, the data was inserted directly into the report. We recorded the data by observing the users’ behavior and listing whether or not they were able to accomplish each task. We also took note of any difficulties they were having, or notable behavior, which is summarized below the table. If a user couldn’t perform a task, we made note of it and helped them through the task.

The results of our initial user testing were:

Task	Installing	Creating A Project	Editing Files	Commit First File	Commit Second File	Programming Background
Student 1	yes	no	yes	yes	yes	**experienced**
Student 2	no	no	yes	yes	no	**novice**
Student 3	no	yes	yes	yes	yes	**novice**

Our biggest observation was that using a terminal was confusing for users. We only require this when installing our project, and we give the exact commands to run, but this feedback shows that we should find a way to avoid using the terminal all together. Additionally, some of our users had trouble with our interface, such as being unable to locate the TyrantVC panel, or finding certain buttons. The very first tasks a user does, such as installing the plugin and creating a repo for their scripts, are the ones that users are least comfortable doing under our current model. We need to revise/rethink how we're asking users to perform these tasks in order to make our plugin more intuitive. Finally, we need to do a better job explaining in the user manual how to perform certain tasks, such as creating a project from a folder, because one user didn't understand what they were being asked to do.

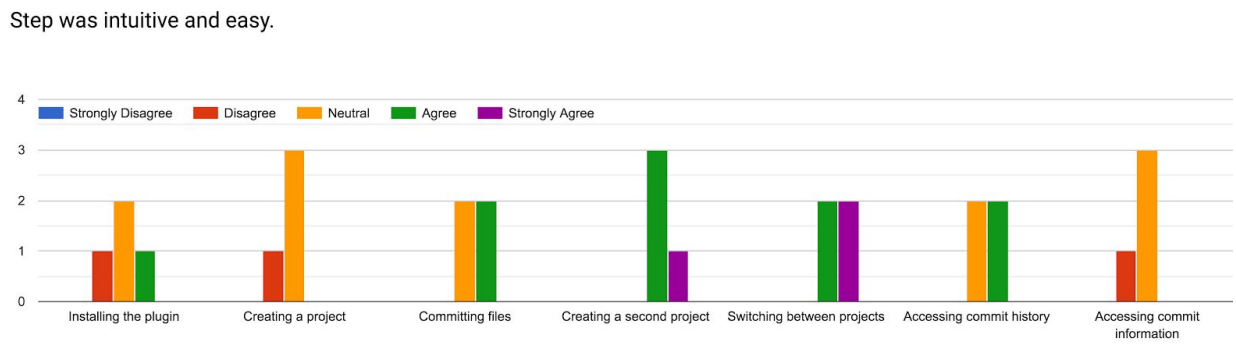
Second Evaluation

The second evaluation was focused on gathering feedback on adjustments that were made to installation process and the new features that were developed since the first evaluation, specifically: interacting with multiple files, looking at commit history of a project and previous versions of the files committed. The users were asked to perform following steps:

1. Visit TyrantVC page (*https://github.com/cagesellchen/TyrantVC*) and install the plugin by following the instructions on the page.
2. Create a project via the plugin from the folder "Project1" on Desktop.
3. Open and modify files in ScriptEditor: *1.py* to print(11), *2.py* to print(22).
4. Commit 1.py with a message "11".

5. Commit 2.py with a message “22”.
6. Create a project via the plugin from the folder “Project2” on Desktop.
7. Open and modify file in ScriptEditor: 3.*py* to print(33).
8. Commit 3.py with a message “33”.
9. Switch back to Project1 and look at the commit history of the project.
10. Switch to Project2 and look at the commit history of the project.
11. Look at the version of the file that was first committed and the contents of that commit.
12. Finally please fill out this google form to tell us about your experience: *https://goo.gl/forms/myWzv21K3TiP3R5F2*

The feedback form included questions about the usability of each portion of the plugin which the users had to answer using a Likert-type scale. The results from 4 user study participants(2 inexperienced, 1 somewhat experienced and 1 experienced) can be seen below (generated via the Google Form results tab).



We were happy to see that the installation process was easier for most of the users, with the only problem arising due to misunderstanding of how to download the installation script. Creating the project still suffered from some vagueness and misunderstanding of what it meant to create a project from an existing folder. Everyone was successfully able to commit files and create more projects which tells us that while our project may suffer in the learnability area it successfully fulfills the efficiency aspect of usability. Finally, accessing the commit history and examining the information there has proven to be intuitive to users familiar with the idea of a version control system, while users who were inexperienced were not sure what information they were looking for. Overall the results were encouraging for most of the features of the plugin

Risks and Payoffs

The main risk of the project is that we have no experience building Maya plugins, and only two of us have experience using Maya at all. For this reason, it is hard to estimate how steep the learning curve will be, and how long it will take to complete our various goals. This could potentially cause us to miss deadlines, and hinders our ability to plan for the future. One way we

can help mitigate this is by consulting with a staff member of the UW animation lab who has experience working with the Maya UI and writing plugins for Maya. Given our inexperience with building Maya plugins, there is also a risk that we will come up against an obstacle (say, a feature that is necessary for our project that is unsupported in Maya's API), which could require a workaround that is overly complicated or sub-par (or both!)

Another risk, like with any version control software, is that we are handling our users files, and problems with our software could potentially cause these files to become lost, inaccessible, or otherwise damaged. This is less likely given that we are using Git to handle most of the actual version control, but it is still a real risk that we should keep in mind.

Application

The target audience for this plugin would be Maya scripters who would like to use Git to manage their scripting files, or Maya scripters who would like some sort of version control but don't have experience with Git or many common programming idioms. Many of the customers that we talked to about the idea for this plugin fell into the second category. Specifically, TyrantVC will primarily be used by Maya scripters who are using Maya for personal or educational purposes rather than as part of their work at a large studio, because large studios likely already have similar tools. This piece of software is meant to be fairly small and easy to understand so it would be possible for someone to build on top of it and modify it for their own needs. We imagine people in schools using this as well, as it would be free and easy to install in a vanilla Maya environment where not many other plugins exist.

The current process for a user trying to script in Maya follows these steps. Let's imagine you are trying to write a script that places some cubes around in the scene using a new UI element. First you would write the scripts in Maya's script editor. You would save them to python files on your computer. You would attempt to run your code in Maya. If your code works the panels show up, if it does not, Maya crashes and any unsaved work is gone. Let's assume your code becomes fairly long and you decide to use an IDE. Now to run your code you have to save everything, open up maya, open up the files there and then run them. If you want to use any Git commands you have to run them outside of Maya, on the files that you've created.

With our plugin the user would get access to intuitive and simplified Git functionality right inside of Maya. As the user works they would be able to save to a repository which they would be able to access from Maya on other devices with this plugin. Saving should feel more reliable and loading up the last iteration of scripts should be quick and easy.

Lessons Learned:

Throughout this project, we learned many valuable lessons and skills. One important takeaway was not to try to force our project to fit the initial design if we discovered along the way that the design was unrealistic or ineffective. For example, our initial UML diagram for splitting up our files included far more files than we needed, and we ended up cutting it down for our final product/diagram. This was mostly because we were new to Maya plugin development, and we didn't realize that certain things should intuitively be together until we were actually coding. There were also files that we hadn't anticipated needing, like config_access. Another takeaway from this project was practicing the ability to quickly familiarize ourselves with a 3rd party API. In this case, that was familiarizing ourselves with the Maya plugin API. We also each gained experience working in a team and using a version control system to contribute to the project. This included experience handling merge conflicts and communicating on who was doing different parts of the project.

Citations

[1] Maya Script Editor,
https://help.autodesk.com/view/MAYAUL/2018/ENU/?guid=GUID-7C861047-C7E0-4780-ACB5-752CD22AB02E

[2] Charcoal Editor, http://zurbrigg.com/charcoal-editor-2

[3] Introduction to Maya Embedded Language (MEL) for Programmers,
https://help.autodesk.com/view/MAYAUL/2018/ENU/?guid=GUID-0F7C50D1-FF45-4868-8EBC-FE044F54B82E

[4] Github Desktop, https://desktop.github.com/

[5] MEL system command,
https://help.autodesk.com/cloudhelp/2016/CHS/Maya-Tech-Docs/Commands/system.html