**CSE 403 Project Proposal**

**Project Name:** TyrantVC
**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. 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.

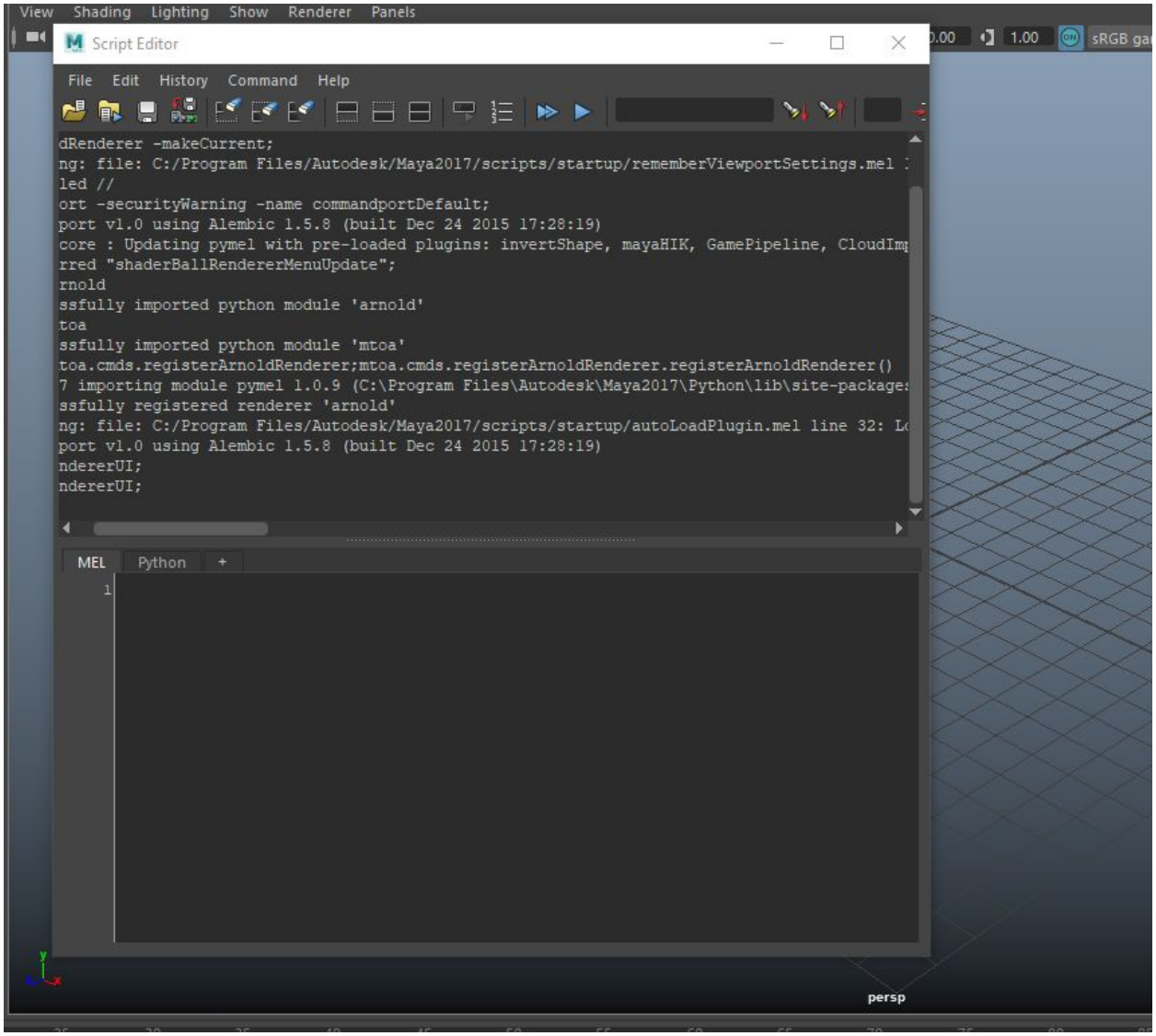


*Fig. 1, a screenshot of Maya's script editor. The top panel is the output, and the bottom panel is where scripts are written.*

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).

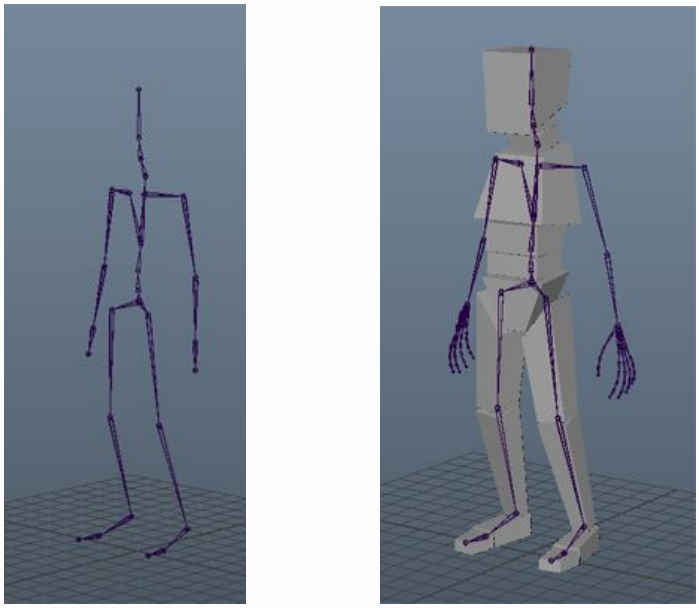

*Fig. 2, visual aid of Maya scripting example -- applying blocks to skeleton.*

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, 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 to pull or push their scripts which results in a messy and inefficient workflow. 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 want to create a plugin for Maya that adds an additional UI panel to the Maya Environment. This UI panel would allow you to see which repository you are working in, which files have been modified, and allow for standard Git functionality of pulling, committing, and pushing files. It will be using Git locally on the user's machine, not GitHub.

**User Interface**

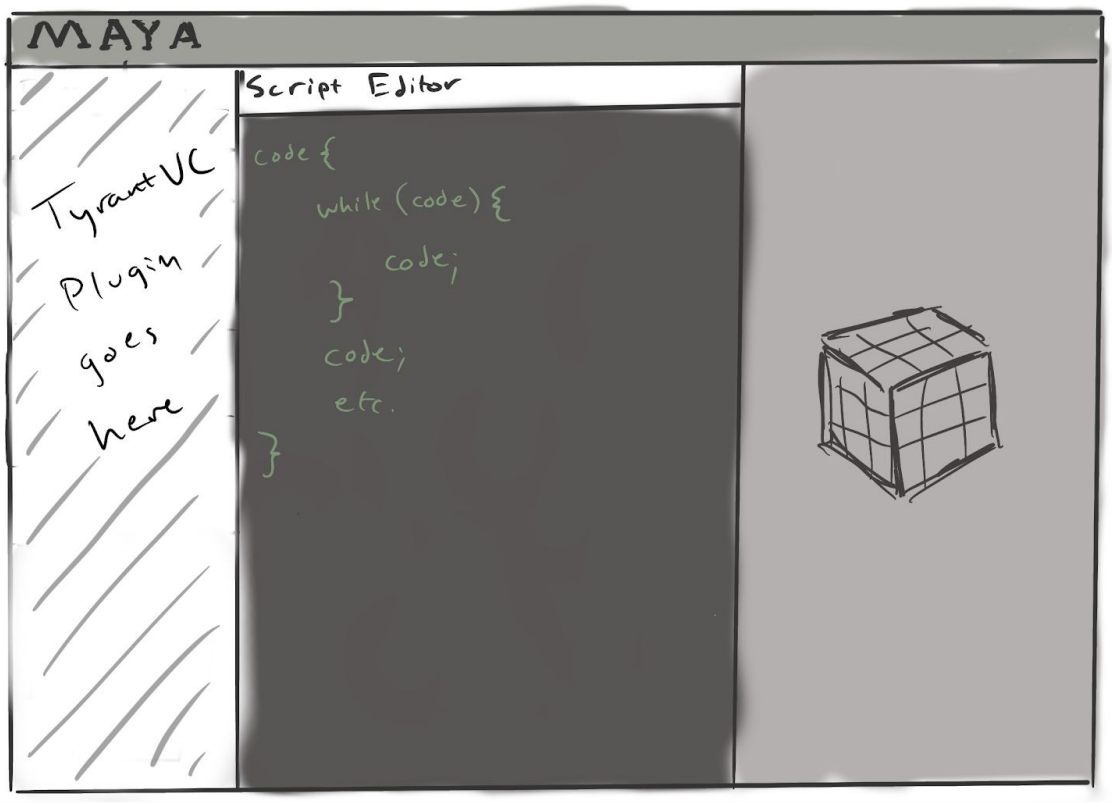


*Fig. 3, basic Maya scripting workspace. The left panel is where the TyrantVC plugin will be located, and the center panel is the normal Maya script editor where developers write their scripts.*

The user will interact with the plugin through a panel that will default to being placed on the side of their Maya workspace. Maya users can rearrange the panels however they see fit, but the diagram above provides an example of what the user's workspace could look like. In the center is the script editor, where their code will be visible and edited, and to the left of that is the panel for the plugin. The plugin will have 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.

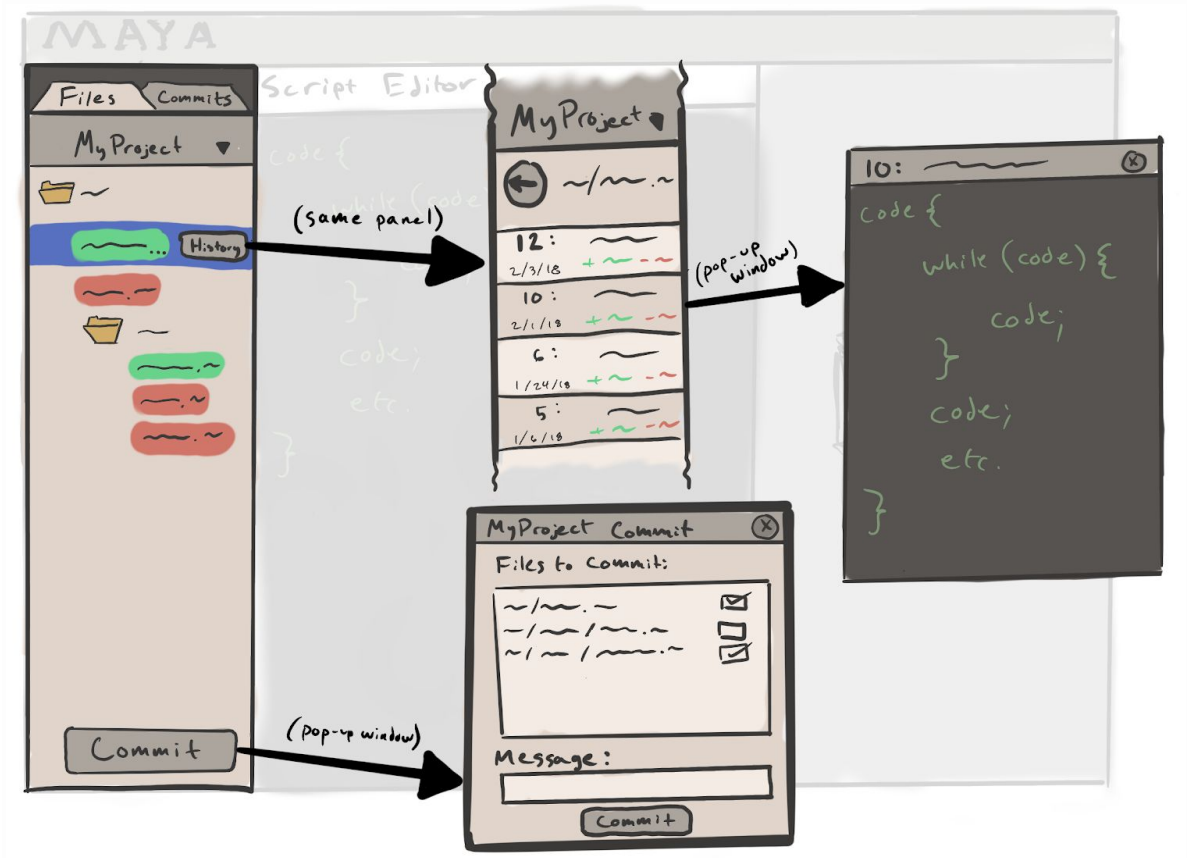


*Fig 4, TyrantVC Files tab UI mockup diagram.*

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. 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.

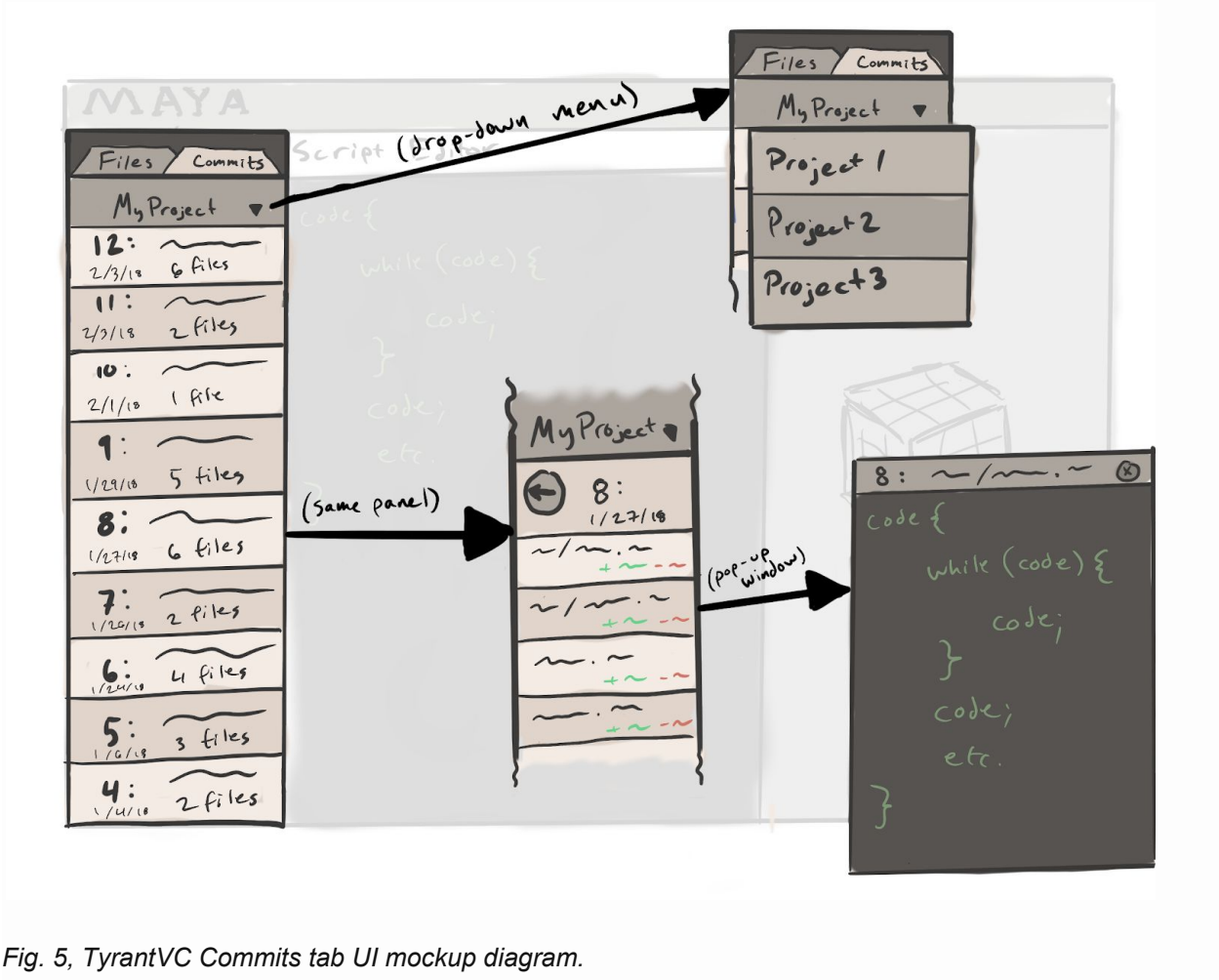


*Fig. 5, TyrantVC Commits tab UI mockup diagram.*

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. Finally, in both tabs, users can click the drop-down menu at the top to change which repository they are working on.

**Implementation Details**

The plugin will be written in Python and tested using Maya and its script editor. The plugin will consist of three major elements: the UI, connection to git, and the connection between the UI code and the git code. The UI component will be 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 will be mostly simplified to stating which repository and branch you are working in, which files have been changed, and buttons for pulling, committing and cloning/making a new repo. The component that will allow the connection to git will be written in Maya's own scripting

language, MEL. This component needs to be tested on various operating systems since access to bash shell in Maya using MEL varies between UNIX based systems and Windows [5]. Finally the component that will connect the two previous pieces together will be written in PyMEL, Maya's python wrapper for MEL. The purpose of this component is to connect the MEL code and PyQT code into a working plugin for Maya.

**Architecture:**

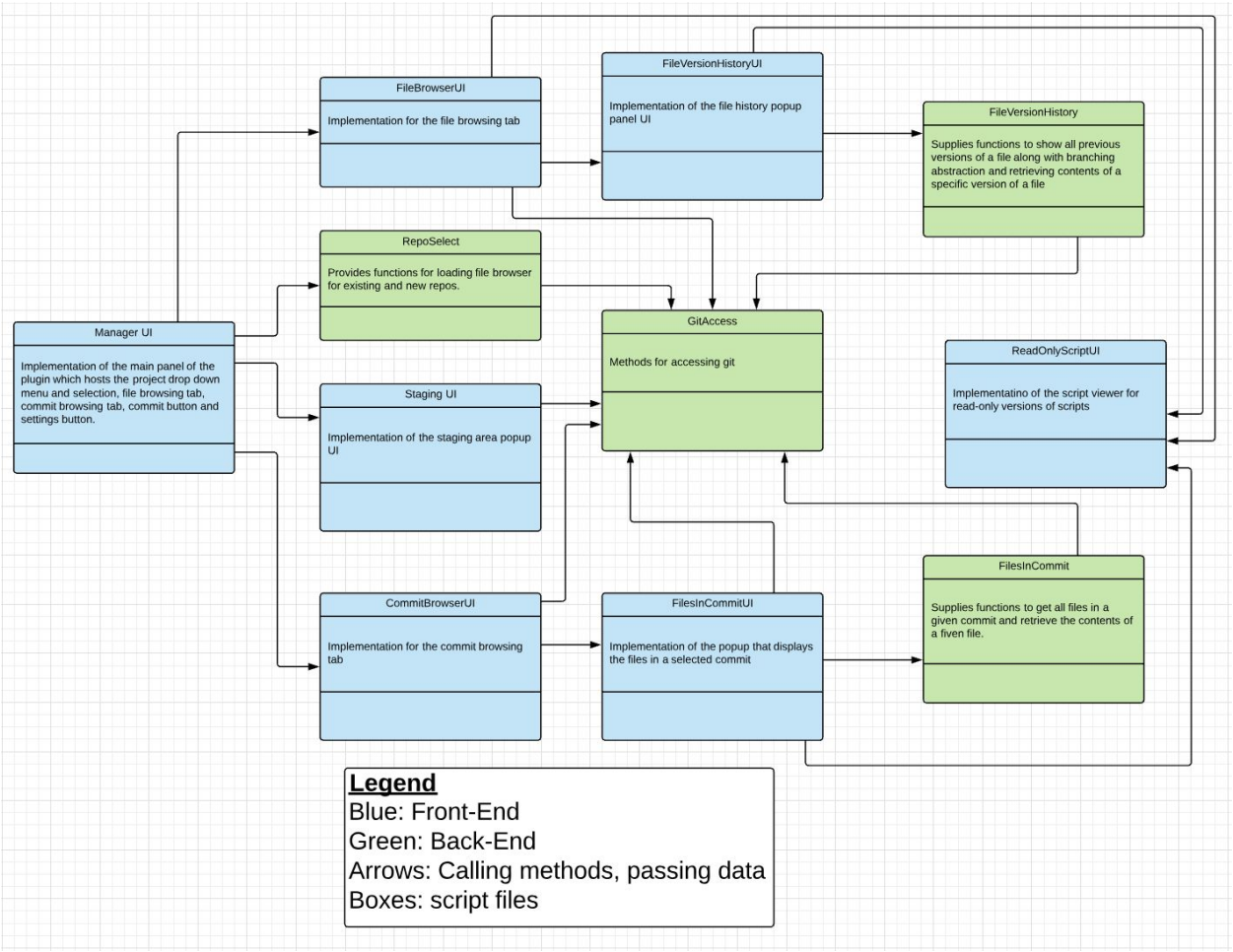


*Fig. 6, TyrantVC code architecture diagram*

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:

- **Manager UI:** This class houses the main panel of our plugin, which hosts the project drop-down menu and selection, the file browsing tab (further abstracted), the commit

browsing tab (further abstracted), the commit button, and the settings button. It calls/communicates with the FileBrowserUI, CommitBrowserUI, StagingUI, and RepoSelect classes to create popups and fill the panel with content based on user interaction.

- **FileBrowserUI:** This class contains the visual implementation for the file browsing tab. Users will be able to view the script files, and all past versions, in their repo. To do this, it will communicate with GitAccess to populate its panel with content. If a file is double clicked, it will initiate a popup script editor by calling the ReadOnlyScriptUI class. If a file is single clicked or right clicked, it will give the option to view all versions of the file. If this is selected, it will initiate a popup by calling FileVersionHistoryUI.
- **CommitBrowserUI:** This class contains the visual implementation for the commit browsing tab. It will allow users to view previous commits by calling GitAccess, and if a user selects a commit entry, it will display the files saved in that commit in a popup box by calling FilesInCommitUI.
- **StagingUI:** This class implements the popup for the git staging area. In this popup, users can view all modified files and select which ones they would like to commit. They can also add a commit message. Once they select the “commit” button, the class calls methods in the GitAccess class to “add/commit/push” the selected files.
- **FileVersionHistoryUI:** This class implements the popup containing a file’s version history. When a user selects a file in the version history, it will initiate a popup box to view the script by calling ReadOnlyScriptUI.
- **FilesInCommitUI:** This class creates a popup displaying the files in a given commit. It does this by calling methods from FilesInCommit. If a file is selected, it displays the file by calling ReadOnlyScriptUI.
- **ReadOnlyScriptUI:** This class creates a popup containing the text of a script file. This only provides reading functionality; the user cannot modify this text, only view and copy it.

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

- **GitAccess:** this class provides methods for all interactions with git. This includes functionality for committing files, viewing previous versions of a file, viewing previous commits, loading existing repos and creating new repos,
- **RepoSelect:** This class interacts with GitAccess to populate the file/commit browser with existing repos, or creating a new repo
- **FilesInCommit:** This class supplies functions for interacting with GitAccess to retrieve a specific commit and load files from that commit.
- **FileVersionHistory:** This class supplies functions for interacting with GitAccess to supply the version history for a given file, including the time and name of each commit which modified that file.

**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 can be tested through comparison to a predefined specification file which would define all of our plugin's functionality. Some of this could be done with automatic testing and some would have to be performed by hand. For example, since all MEL commands produce some output, we could 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 by hand. 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 or not sufficient enough functionality. 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.

**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.

**Schedule**

Week 4 (1/29):

Finish project proposal. Interview Maya users to draft up the specification file for our plugin listing key elements of its functionality, as well as a list of “nice-to-have” features

Week 5 (2/5):

Test each of the technologies needed for our project, and create a detailed architecture of the program and how these technologies interact. Write a manual for our program.

Week 6 (2/12):

Front end: Have simple Maya plugin working, which displays a disconnected interface for the program’s intended functionality

Back end: Implement the functions required for communication with Github, have a basic interface for the front end to interact with

Week 7(Midterm, 2/19):

Connect front end and back end, to create a usable version of our program (functional, but unpolished). Main feature of commiting files to repo should be working by this week.

Week 8 (2/26):

Conduct user tests on with at least 3 Maya users:

- Test their ability to understand the program without guidance

Given a request to fulfill some action, observe how they easily they can carry it out

Interview the user about their experience, what their difficulties were, and what features they would most likely use in their work.

From this, compile detailed list of their feedback and newly discovered issues with our software. Come up with ways to potentially fix or restructure aspects of the program to improve their experience.

Week 9 (3/5):

Complete about half of the bug fixes/redesigns from Week 8, once ready, get more feedback from Maya users.

Week 10 (Final, 3/12):

Have a finished product which implements all of the specified functionality (taking into account feedback from user tests), and has an interface which is easy to use.  Implement nice to have features, if time allows.

**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

**Feedback**

We have addressed all feedback.