

Pogil Plugin

Eli Lewis and Yigit Turan

Professor Victor Norman, Project Advisor

Fall 2024 Senior Project

Github: <https://github.com/PogilPlugin>

Problem Description

The issue we are attempting to address with the development of the POGIL plugin is one of accessibility. Currently, POGIL activities can be created manually or by use of a Google Docs extension. However, Microsoft Word has long been the dominant word processing program in both Business and Academia. By developing a Microsoft Word plugin which allows the generation of POGIL activities, we are hoping to allow for further access to the POGIL process and the benefits it holds, while simultaneously alleviating the burden from professors of having to use software they may not be familiar with (Google Docs).

Background

POGIL, or Process Oriented Guided Inquiry Learning, is a teaching method where students are split into small groups and assigned roles to complete a worksheet. In order to efficiently create these worksheets, Clif Kussmaul created a Google Docs extension which allowed an author of a POGIL activity to create one "template" file, with instructions, questions, and answers, and notations, and then from that template file, create separate "Teacher" and "Student" versions.

Solution

Our solution to this problem was very simple; We proposed, designed and created a plugin which directly integrates with MS Word and allows users to create POGIL documents while taking advantage of all of Word's features.

Design Norms

We believe that these design norms are apparent in our work:

- 1) Cultural Appropriateness
- 2) Caring
- 3) Transparency & Aesthetics

Firstly, our work is culturally appropriate because it relieves the burden of manually creating POGIL activities (or utilizing Google Docs, when unfamiliar with the program) while maintaining the benefits of using MS Word, such as the

formatting capabilities. Secondly, we believe that our work is caring, while it helps us serve and love our neighbors (fellow students and professors) while not enabling any foreseeable harm. Finally, we believe that our work aligns with the norms of transparency and aesthetics, because it follows the design language of Microsoft Word and is simple, intuitive, and user friendly.

Development Approach

We utilized an AGILE like project management approach to developing this program. Every week, we met with our advisor and were assigned specific “tasks” (AKA stories) which we would then complete in the next week (sprint) before meeting again and either continuing or pivoting based on the progress made and discoveries uncovered. We utilized Git and GitHub as version control. We also employed a type of code review system between ourselves, and would often look over the others code to ensure that it met our expected standards.

Problems Encountered and Overcome

We faced the following challenges throughout the development process of this project:

- 1) We had difficulty publishing the extension.
- 2) We had issues converting the file to a PDF.
- 3) We encountered troubles parsing the XML and dealing with special symbols and images.

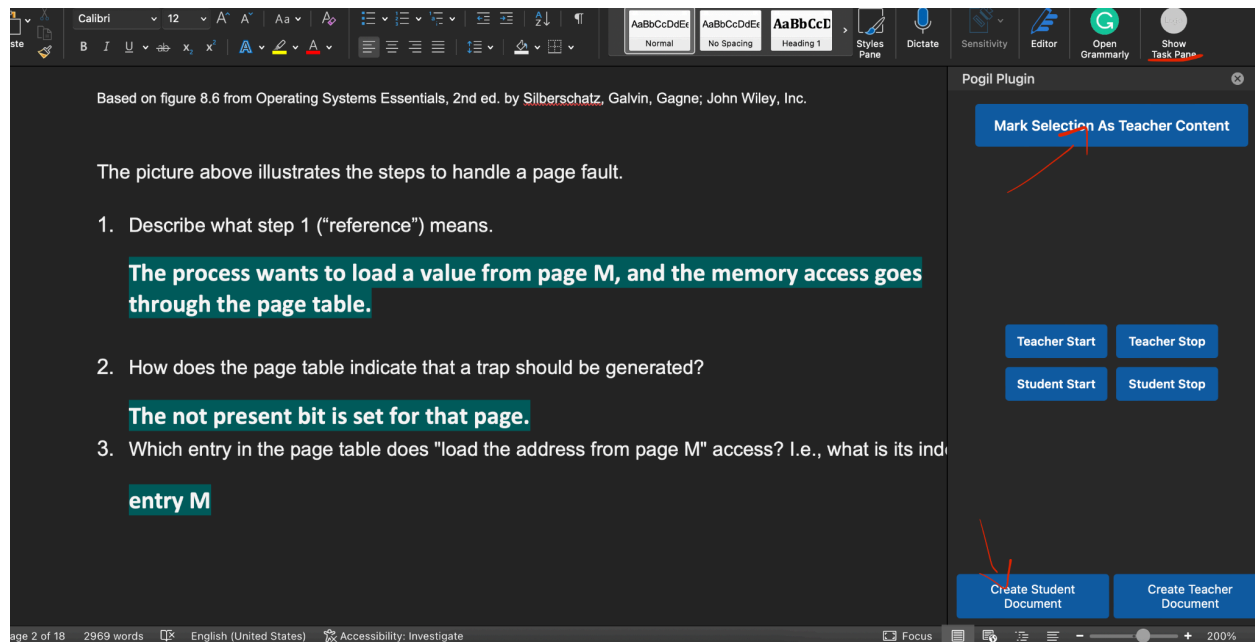
Sadly, we never managed to figure out how to publish the MS Word plugin. However, we met with Professor Norman, and came to the conclusion that he could speak with Professor Arnold and figure out the proper way to publish the plugin. Secondly, originally we had hoped to allow the generation of both .docx and .pdf files with our extension. Unfortunately, we had some difficulties with keeping the results consistent across file formats. Eventually however, we realized that we shouldn't try to create a .pdf document at all, and instead leveraged the internal features of MS Word; after creating a .docx file, the professor can simply export to .pdf from Word. This allows for more consistent results and keeps the design simple. Finally, we had some issues parsing the XML of the Word documents and dealing with specialty symbols. We surpassed this issue when, during a casual

conversation with another professor about our project, they suggested we try an alternative method for parsing the XML (running it as a tree). This, along with the lucky coincidence that we had recently encountered the DOMParser class in Javascript, gave us the insight to fix all of our parsing issues.

Testing Methodology

We tested our project extensively during our development process through two main methods. Firstly, we acquired a POGIL document from professor Norman. Then we manually tested the extension, utilizing it as a user might, in order to guarantee proper functionality and to get a feel for how it functions. We also manually went through the XML document of the Word document and ensured that the structure looked consistent with the specification of a word document, even after our edits had taken place. We deemed this testing sufficient for the relatively simple program we designed.


Demo



Mark selection as teacher content in source document

Based on figure 8.6 from Operating Systems Essentials, 2nd ed. by Silberschatz, Galvin, Gagne; John Wiley, Inc.

The picture above illustrates the steps to handle a page fault.

1. Describe what step 1 ("reference") means.
2. How does the page table indicate that a trap should be generated?
3. Which entry in the page table does "load the address from page M" access? I.e., what is its index?
4.  What does step 5 ("reset page table") mean? What two things does the OS change

After pressing generate student document, the selected content is removed

Future Work

In the future we hope that the extension might be published to the MS Word store so that other professors can easily use it. This would also create the opportunity to generate user feedback that could allow the plugin to be iterated on further in another senior project.

Acknowledgements

We would like to thank our project advisor, professor Norman, for all the help that he gave to us. We would also like to thank Clif Kussmal for creating the Google Docs POGIL Extension which inspired this project.