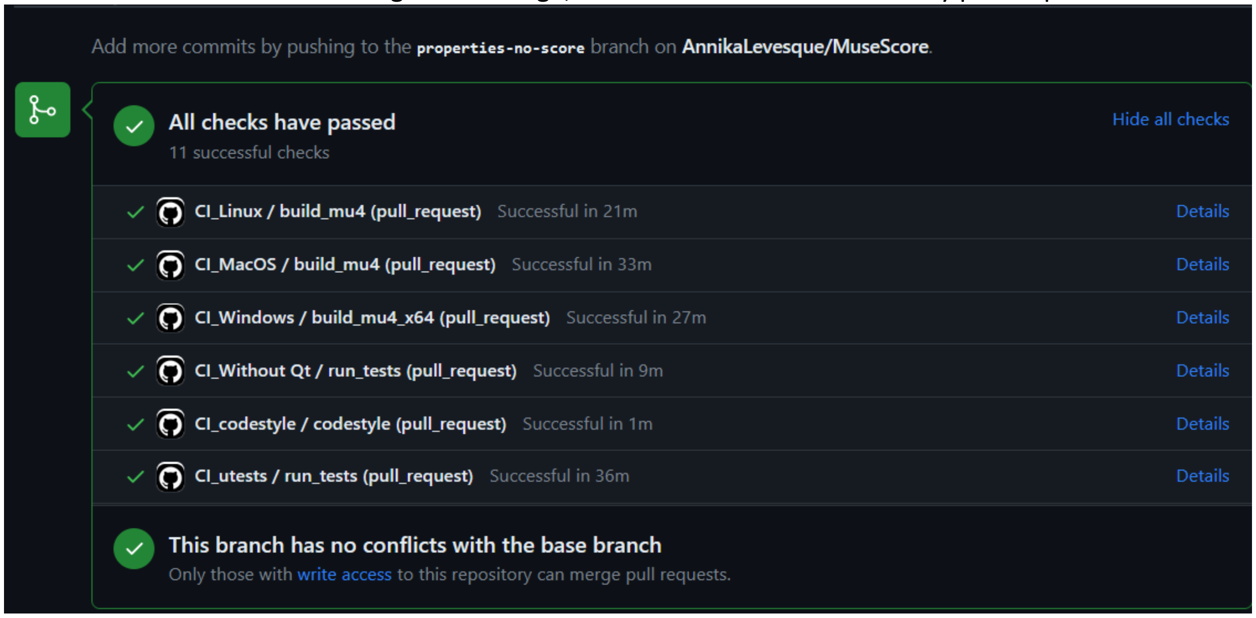
Homework 6b

Jing-Yi Liu (jingyiyl@umich.edu) (no collaborators)

1. **Selected project:** I selected the MuseScore Project (GitHub: https://github.com/musescore/MuseScore, website: https://musescore.org/en) which is an open-source music notation software. In other words, it allows composers to typeset sheets of music in the same way Microsoft Word, for example, allows writers to typeset documents.
2. **Social Good indication:** I do not believe that MuseScore contributes to the 17 Sustainable Development Goals. However, it is one of the only (and as far as I know, the first) completely free, professional, modern music notation software, so it helps reduce the barrier to entry for musicians and composers and reduce inequality in music education.
3. **Project context**[1,2,3,4]**:** MuseScore started as a free open-source program developed by 3 people (Werner Schweer, Nicolas Froment and Thomas Bonte), with the support of the larger music community. It is released under the GNU General Public License, ensuring it will always be free and open source. As the founders put it, the mission was to "democratise access to sheet music". It is now the most popular notation software on the market, and was acquired by Ultimate Guitar in 2017, who contributed full-time paid developers to the team, including Martin Keary (Lead Designer) and Vasily Pereverzev (Lead Developer). It seems like Schweer, Froment, and Bonte are no longer actively involved in development. Development is paid for by the sheet music sharing platform musescore.com (the software website is musescore.org), in which users can upload and download sheet music made by the community. Accessing the catalog of scores (which also includes official scores licensed from famous music publishers) requires paying an annual subscription of $39.99. As of the time of writing, the only major competitors to MuseScore are Dorico ($579.99), Finale ($600), and Sibelius ($9.99/month), none of which are accessible to beginner and amateur musicians. The other free alternatives like Noteflight and Flat.io have severely limited capabilities in comparison to MuseScore, since their business model locks functionalities behind paywalls, unlike MuseScore, where the entire program with all its functionalities is free and open source. With the update to MuseScore4, its developers have also created the state-of-the-art playback technology MuseSounds (hear the difference here: https://youtu.be/Qct6LKbneKQ?t=1858), and as far as I know is the only notation software currently with this technology built in (previously, musicians had to manually program Dorico/Sibelius/etc. to work with NotePerformer or some other technology to get the same effect, something that takes tremendous technical skill). In my opinion, MuseScore is no longer just "the free one" when compared to other notation software, but is genuinely the best music notation software on the market.
4. **Project governance:** All contributions are done through the GitHub repository, and formal communication about a specific issue or pull request are done on those respective pages on GitHub. General/informal discussion about development are done on the forum at https://musescore.org/en/forum/687, and on the discord server at https://discord.gg/HwHhXEbJ4r. News from the project maintainers to the community is (more informally) distributed through the form, (more formally) through the blog at https://musescore.org/en/news and on various social media, and (rarely) through the YouTube channel of the lead designer: https://www.youtube.com/@Tantacrul. On the GitHub page, there is a wiki (https://github.com/musescore/MuseScore/wiki/) with information on contribution,

[1] https://blog.musescore.com/post/171048706627/musescore-joins-ultimate-guitar
[2] https://www.youtube.com/watch?v=Qct6LKbneKQ
[3] https://en.wikipedia.org/wiki/MuseScore#History
[4] https://blog.musescore.com/post/171048786232/ultimate-guitar-welcomes-musescore

including a set of guidelines about style and coding principles that contributors must follow (https://github.com/musescore/MuseScore/wiki/CodeGuidelines). The source code also includes a test suite, and the wiki includes an entire section on suggested testing practices. There is also a wiki section on functional requirements (https://github.com/musescore/MuseScore/wiki/Functional-requirements), but it seems to be still in progress. The wiki also describes how contributions are to be made; how to fork the repository, make feature branches, pull requests, etc. (https://github.com/musescore/MuseScore/wiki/Submit-a-Pull-Request). Every pull request made is automatically run through 11 continuous integration tests (which include style checks) to make sure the contribution is good to merge; see below for the checks on my pull request.



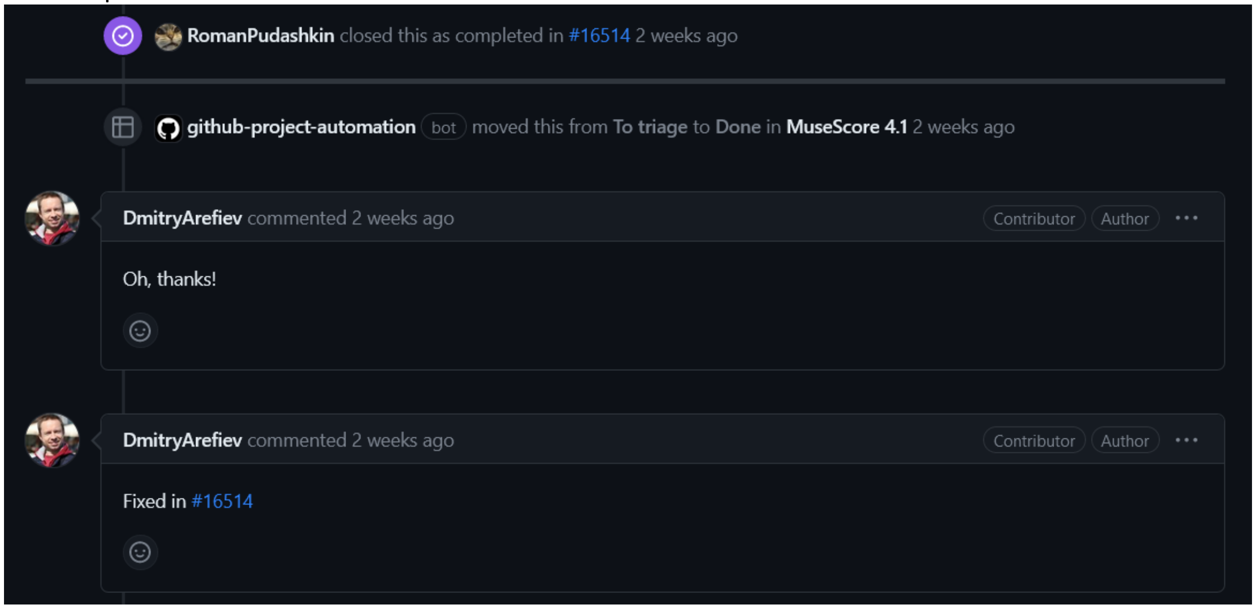
Add more commits by pushing to the `properties-no-score` branch on **AnnikaLevesque/MuseScore**.

All checks have passed
11 successful checks
Hide all checks

✓ **CI_Linux / build_mu4 (pull_request)** Successful in 21m Details
✓ **CI_MacOS / build_mu4 (pull_request)** Successful in 33m Details
✓ **CI_Windows / build_mu4_x64 (pull_request)** Successful in 27m Details
✓ **CI_Without Qt / run_tests (pull_request)** Successful in 9m Details
✓ **CI_codestyle / codestyle (pull_request)** Successful in 1m Details
✓ **CI_utests / run_tests (pull_request)** Successful in 36m Details

This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

Pull requests must be reviewed and accepted by a project maintainer (a.k.a. internal developer, i.e. employee at Ultimate Guitar) before they are merged into the master branch. See below for an example closed issue.



RomanPudashkin closed this as completed in #16514 2 weeks ago

github-project-automation bot moved this from **To triage** to **Done** in **MuseScore 4.1** 2 weeks ago

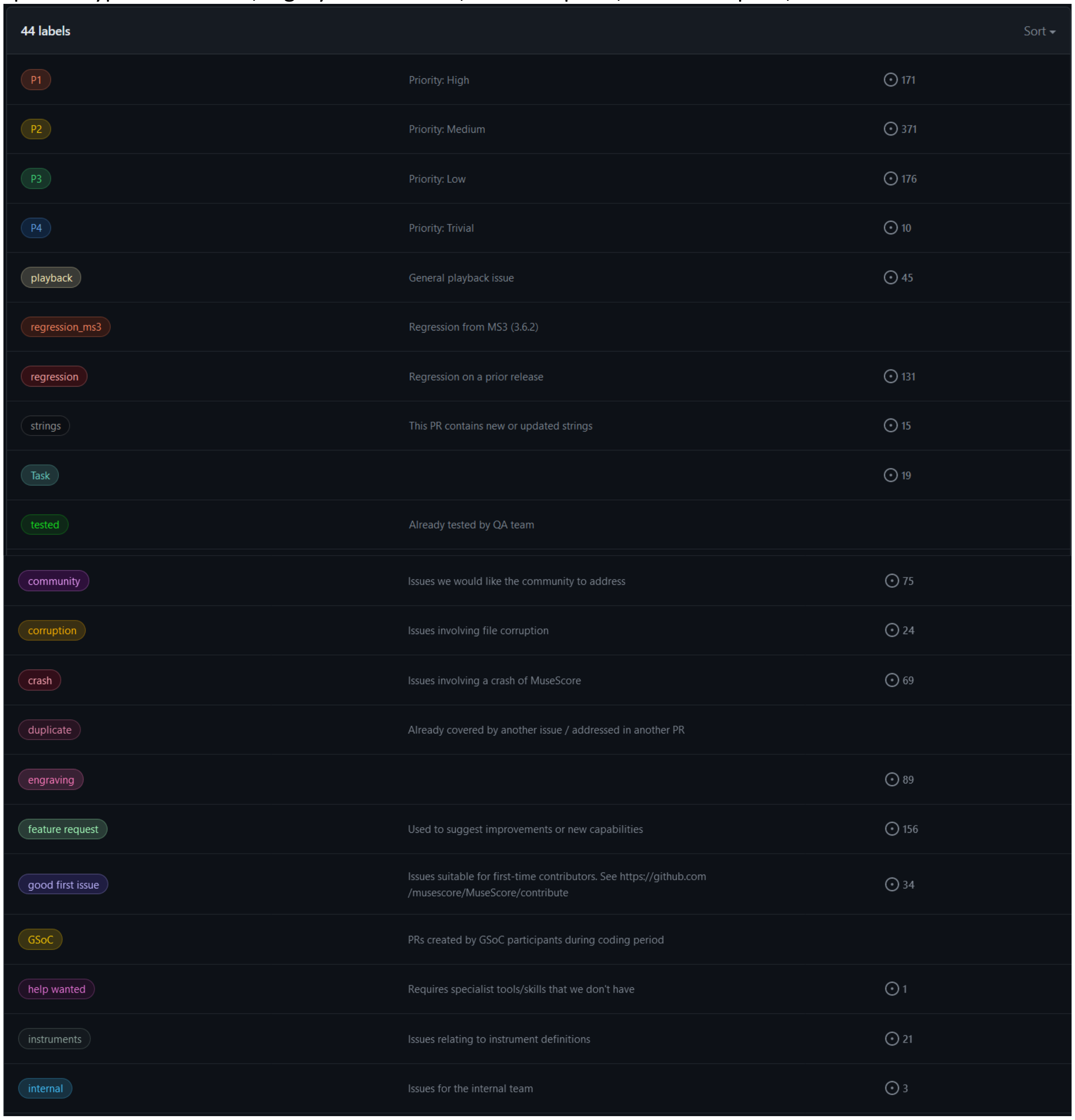
DmitryArefiev commented 2 weeks ago Contributor Author

Oh, thanks!

DmitryArefiev commented 2 weeks ago Contributor Author

Fixed in #16514

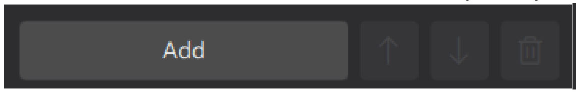
Issues are labeled by priority and severity, as well as the type of issue (such as engraving, playback, UI, etc.). The priority system has 4 levels and the severity system is based on the specific type of behavior; e.g. system crashes, file corruption, feature request, etc.

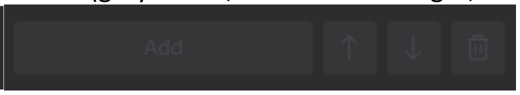


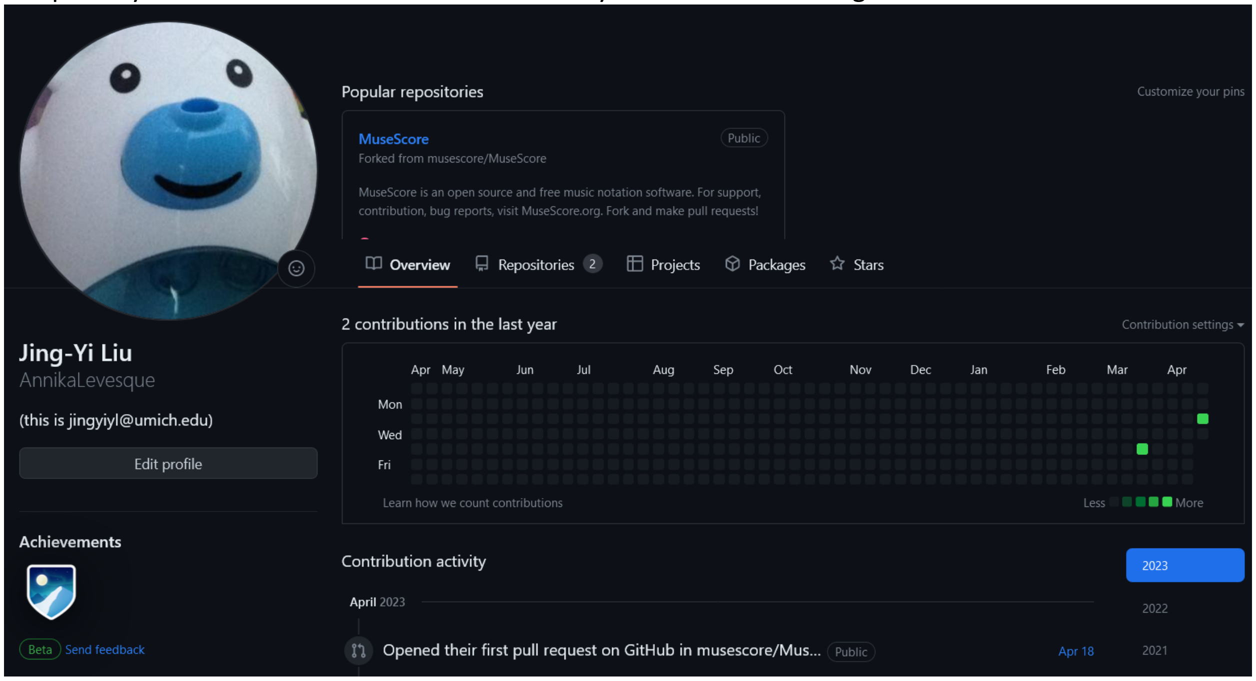
44 labels Sort

P1	Priority: High	171
P2	Priority: Medium	371
P3	Priority: Low	176
P4	Priority: Trivial	10
playback	General playback issue	45
regression_ms3	Regression from MS3 (3.6.2)	
regression	Regression on a prior release	131
strings	This PR contains new or updated strings	15
Task		19
tested	Already tested by QA team	
community	Issues we would like the community to address	75
corruption	Issues involving file corruption	24
crash	Issues involving a crash of MuseScore	69
duplicate	Already covered by another issue / addressed in another PR	
engraving		89
feature request	Used to suggest improvements or new capabilities	156
good first issue	Issues suitable for first-time contributors. See https://github.com/musescore/MuseScore/contribute	34
GSoC	PRs created by GSoC participants during coding period	
help wanted	Requires specialist tools/skills that we don't have	1
instruments	Issues relating to instrument definitions	21
internal	Issues for the internal team	3

There are also tags that indicate an issue is intended for the community to address vs the internal development team, issues that require specialist tools/skills, and good first issues for new contributors. Some issues are assigned to specific internal developers, especially the ones that need to be addressed quickly, but in general there is no system for assigning or claiming specific issues. There is an internal QA team, but their actions are not very transparent to the public. In general, the project maintainers and wider community are very active (I've always gotten responses for any questions in the Discord or on GitHub within a few hours), so the communication feels informal yet rigorous.

5. **Task description:** I implemented task 1 from HW6a: https://github.com/musescore/MuseScore/issues/8964. In essence, there are buttons in the UI that only have functionality when the user has a score open, and a member of the community wanted those buttons to be disabled or removed when there was no score open (analogous to having Microsoft Office open without any particular document or file open). There was already a previous attempt at implementing this by making the buttons unresponsive when no score was open (https://github.com/musescore/MuseScore/pull/13584) but it was rejected because the desire was to have the buttons be completely disabled (greyed out, as seen to the right, below)




or removed, not just have them unresponsive (i.e. the animation that indicates a button is pressed doesn't play). I started my implementation by trying to modify the previous attempt to remove the buttons instead of simply making them unresponsive, but I soon realized that the previous attempt modified each button one-by-one, but the buttons in question were grouped together in a module in the code. It would be faster for me (and better for code readability/maintainability) if I changed the visibility of the entire module. The implementation of the panel is such that it is "rebuilt" every time the user changes their selection (because the panel contains options regarding the item being selected), so my implementation prevents the modules in question from being built when there is no score open.
6. **Submitted artifacts:** My official pull request is at https://github.com/musescore/MuseScore/pull/17278. Below is a screenshot of my profile, which is at https://github.com/AnnikaLevesque. The username is for privacy reasons but I have temporarily added identification information to my account for this assignment.

Popular repositories

Customize your pins

MuseScore — Public

Forked from musescore/MuseScore

MuseScore is an open source and free music notation software. For support, contribution, bug reports, visit MuseScore.org. Fork and make pull requests!

Overview · Repositories 2 · Projects · Packages · Stars

Jing-Yi Liu

AnnikaLevesque

(this is jingyiyl@umich.edu)

Edit profile

2 contributions in the last year

Contribution settings

Learn how we count contributions

Achievements

Beta · Send feedback

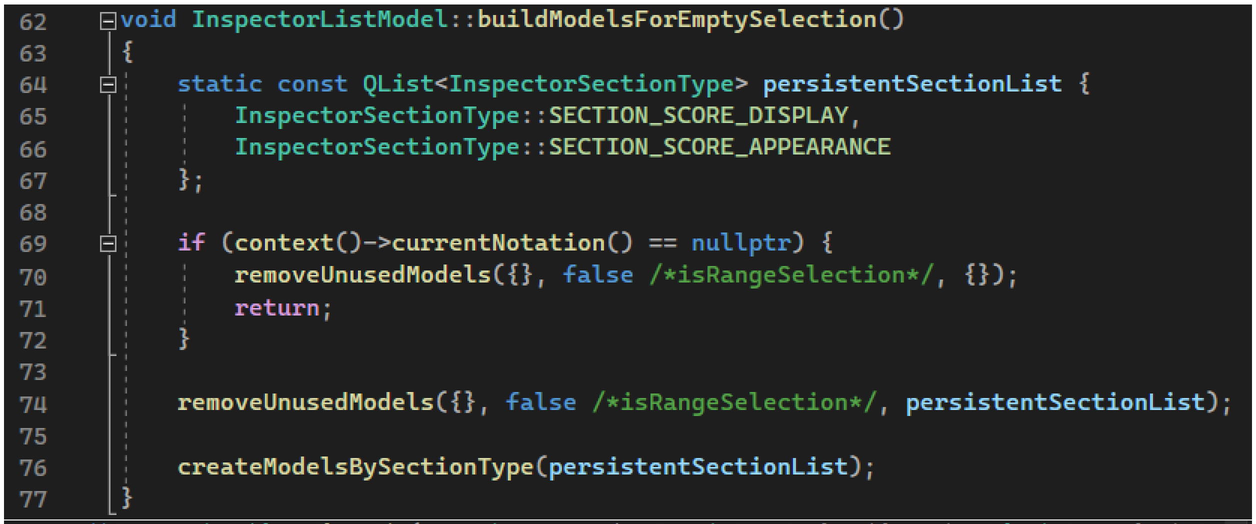
Contribution activity

April 2023

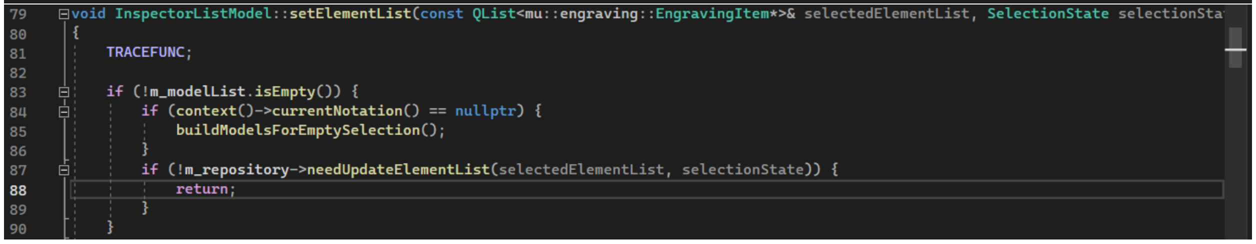
Opened their first pull request on GitHub in musescore/Mus... Public — Apr 18

2023 · 2022 · 2021

Screenshots of the code involved is below: (I added lines 69-72 on the first image and lines 84-86 on the second image)

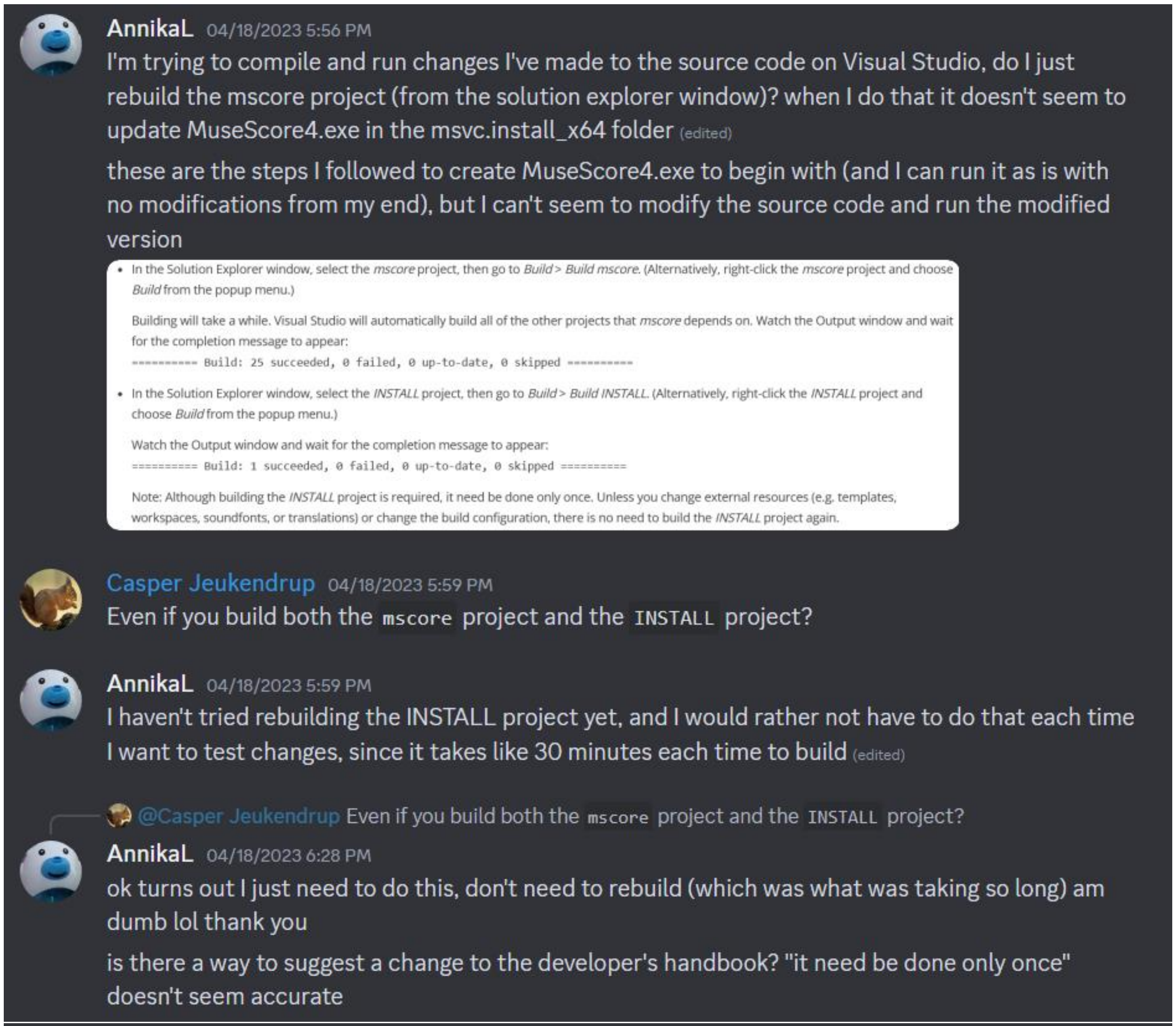


```
void InspectorListModel::buildModelsForEmptySelection()
{
    static const QList<InspectorSectionType> persistentSectionList {
        InspectorSectionType::SECTION_SCORE_DISPLAY,
        InspectorSectionType::SECTION_SCORE_APPEARANCE
    };

    if (context()->currentNotation() == nullptr) {
        removeUnusedModels({}, false /*isRangeSelection*/, {});
        return;
    }

    removeUnusedModels({}, false /*isRangeSelection*/, persistentSectionList);

    createModelsBySectionType(persistentSectionList);
}
```



```
void InspectorListModel::setElementList(const QList<mu::engraving::EngravingItem*>& selectedElementList, SelectionState selectionSta
{
    TRACEFUNC;

    if (!m_modelList.isEmpty()) {
        if (context()->currentNotation() == nullptr) {
            buildModelsForEmptySelection();
        }
        if (!m_repository->needUpdateElementList(selectedElementList, selectionState)) {
            return;
        }
    }
```

the object context()->currentNotation() is a pointer to the score that the user has opened, so if it is nullptr, that means that no score is open. Each Model is an aforementioned module in the panel, and the persistentSectionList is the list of modules that was previously in the panel when no element in the score was selected (whether or not a score was open). My modifications made it so that when the panel is built for "EmptySelection" (i.e. the user isn't selecting anything like a note or a piece of text to edit their properties), it first checks if a score is open, and if not, the panel is emptied. Also, since the panel is only updated when the user's selection changes (e.g. when the user goes from selecting something to not selecting anything), I had to add a check for if the user closes the score while not selecting anything (because in that edge case, the user's selection goes from nothing to nothing-- there's no change, even though the program went from score open to no score open). I didn't write any test cases because my modification was a UI change, but I did run the included test suite to make sure I didn't break anything, and the results are in the appendix of this report. I also asked the community for help in the official discord when I ran into issues (see below):



AnnikaL 04/18/2023 5:56 PM

I'm trying to compile and run changes I've made to the source code on Visual Studio, do I just rebuild the mscore project (from the solution explorer window)? when I do that it doesn't seem to update MuseScore4.exe in the msvc.install_x64 folder (edited)

these are the steps I followed to create MuseScore4.exe to begin with (and I can run it as is with no modifications from my end), but I can't seem to modify the source code and run the modified version

- In the Solution Explorer window, select the *mscore* project, then go to *Build > Build mscore*. (Alternatively, right-click the *mscore* project and choose *Build* from the popup menu.)

  Building will take a while. Visual Studio will automatically build all of the other projects that *mscore* depends on. Watch the Output window and wait for the completion message to appear:

  ```
  ========== Build: 25 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
  ```

- In the Solution Explorer window, select the *INSTALL* project, then go to *Build > Build INSTALL*. (Alternatively, right-click the *INSTALL* project and choose *Build* from the popup menu.)

  Watch the Output window and wait for the completion message to appear:

  ```
  ========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
  ```

  Note: Although building the *INSTALL* project is required, it need be done only once. Unless you change external resources (e.g. templates, workspaces, soundfonts, or translations) or change the build configuration, there is no need to build the *INSTALL* project again.

Casper Jeukendrup 04/18/2023 5:59 PM

Even if you build both the `mscore` project and the `INSTALL` project?

AnnikaL 04/18/2023 5:59 PM

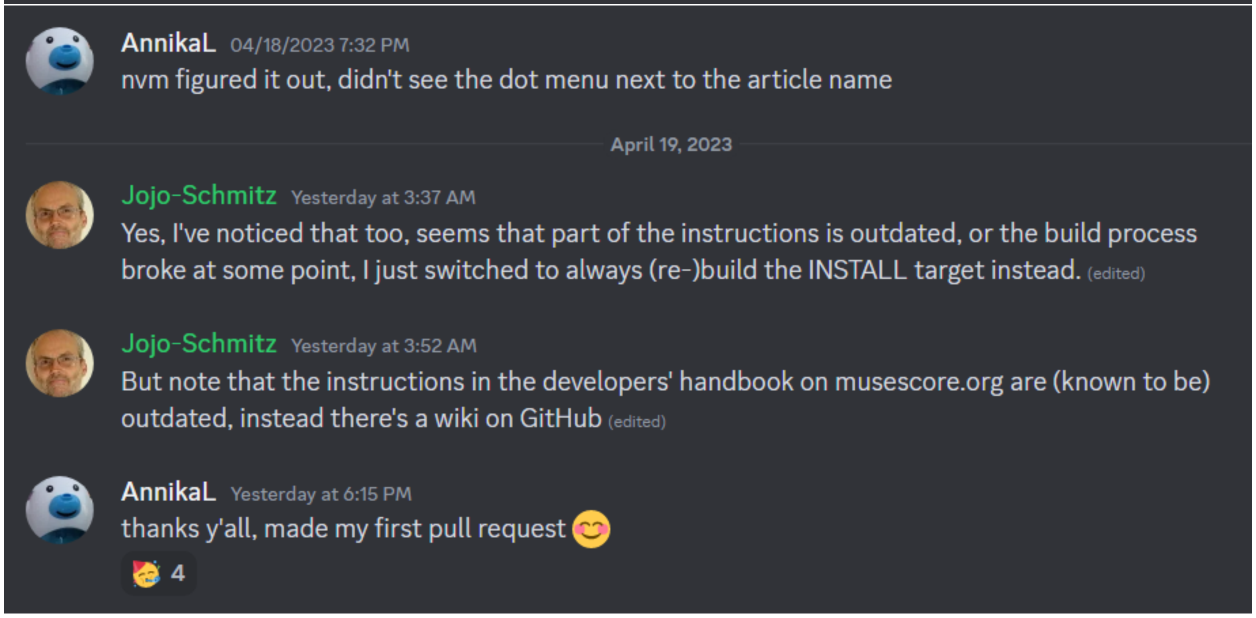
I haven't tried rebuilding the INSTALL project yet, and I would rather not have to do that each time I want to test changes, since it takes like 30 minutes each time to build (edited)

@Casper Jeukendrup Even if you build both the `mscore` project and the `INSTALL` project?

AnnikaL 04/18/2023 6:28 PM

ok turns out I just need to do this, don't need to rebuild (which was what was taking so long) am dumb lol thank you

is there a way to suggest a change to the developer's handbook? "it need be done only once" doesn't seem accurate



AnnikaL 04/18/2023 7:32 PM

nvm figured it out, didn't see the dot menu next to the article name

April 19, 2023

Jojo-Schmitz Yesterday at 3:37 AM

Yes, I've noticed that too, seems that part of the instructions is outdated, or the build process broke at some point, I just switched to always (re-)build the INSTALL target instead. (edited)

Jojo-Schmitz Yesterday at 3:52 AM

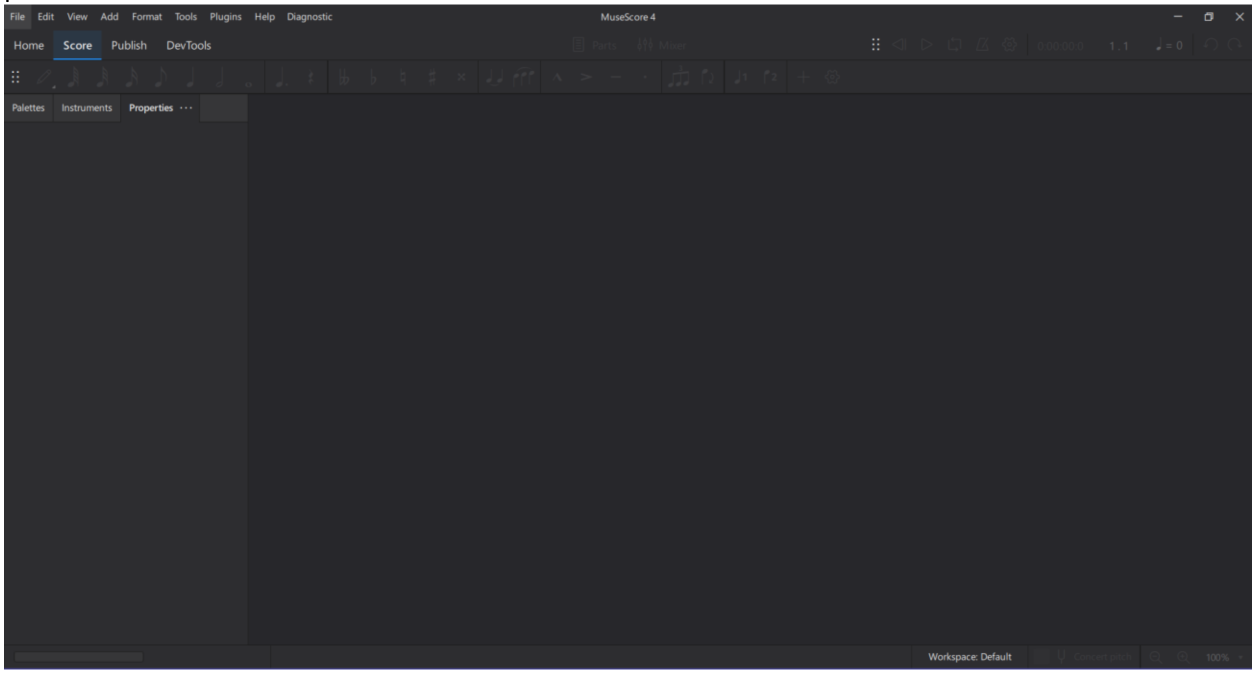
But note that the instructions in the developers' handbook on musescore.org are (known to be) outdated, instead there's a wiki on GitHub (edited)

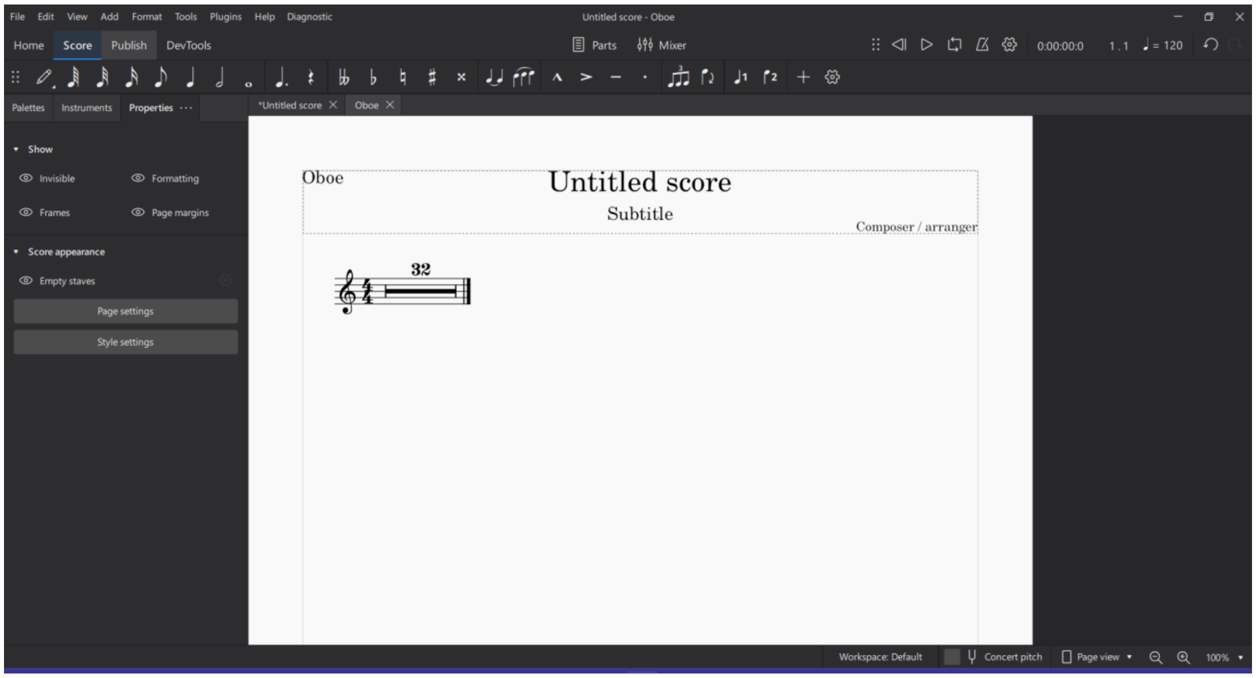
AnnikaL Yesterday at 6:15 PM

thanks y'all, made my first pull request 😊

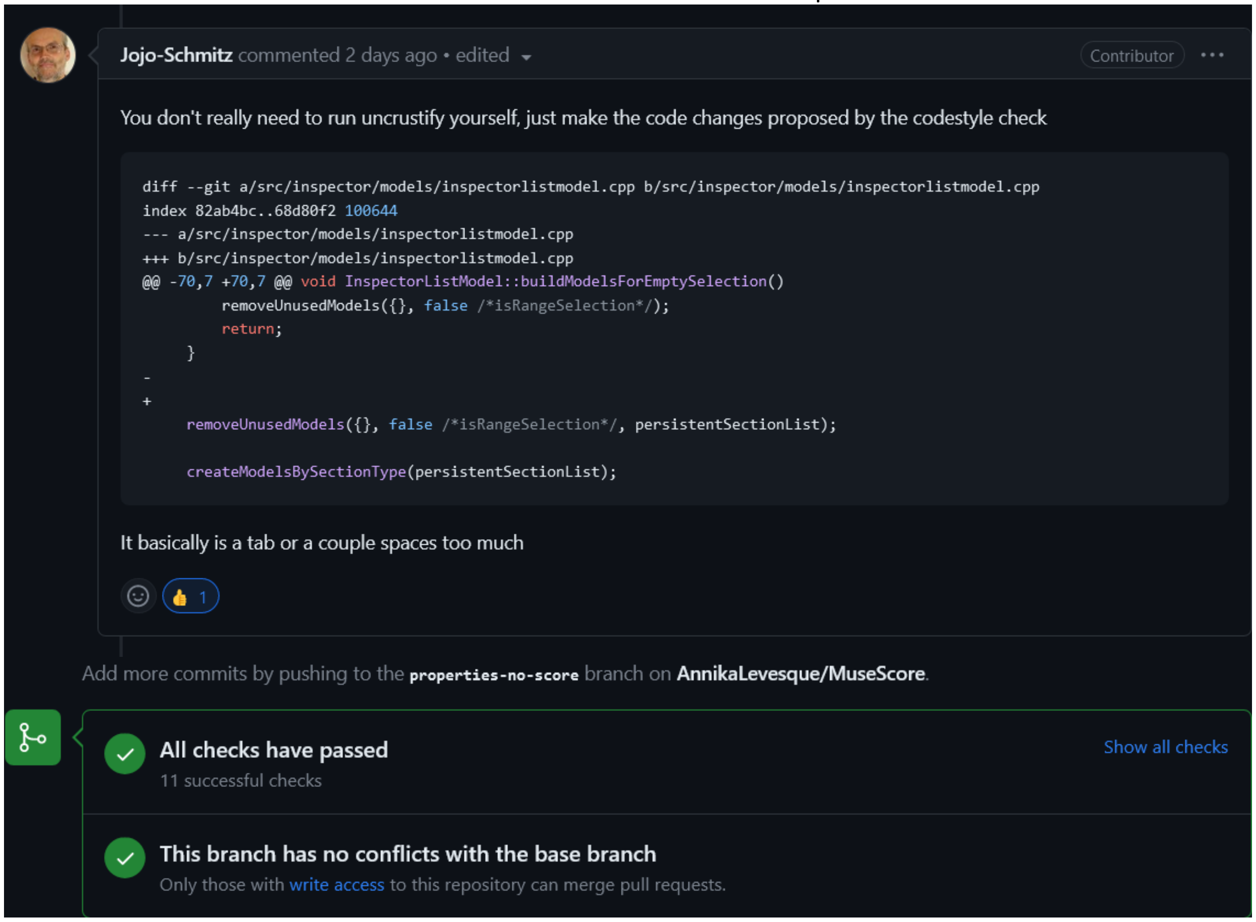
🥳 4

7. **QA strategy:** Since my implementation was a UI change, the bulk of my QA was running the program under various circumstances and making sure the UI behaved as expected. I scoured the tutorials/wiki/program itself to make sure there was no "tutorial" or "onboarding" process that would break if I removed the modules from the panels, then I tried various configurations of opening scores, closing scores, opening multiple scores and closing them one by one vs all at once, having parts open, etc. I paid attention to the specific behavior of the panel, but also that there wasn't any lag when opening/closing scores because I don't want my change to noticeably impact the performance of the program. I also ran the test suite that came with the source code. I also counted on the project maintainers doing an informal code review of my implementation because I know that they are very fastidious about these things.
8. **QA evidence.** Below are screenshots from my manual testing. Note the differences in the left panel.





The output of the built-in test suite is in the appendix. The "failed" tests were all unrelated to the file that I had modified, so I felt it was safe to assume they were not a result of my code. My first attempt passed the continuous integration tests except for the style check, so a project maintainer let me know what the issue was and I fixed it on a subsequent commit:



Jojo-Schmitz commented 2 days ago • edited

Contributor

You don't really need to run uncrustify yourself, just make the code changes proposed by the codestyle check

```
diff --git a/src/inspector/models/inspectorlistmodel.cpp b/src/inspector/models/inspectorlistmodel.cpp
index 82ab4bc..68d80f2 100644
--- a/src/inspector/models/inspectorlistmodel.cpp
+++ b/src/inspector/models/inspectorlistmodel.cpp
@@ -70,7 +70,7 @@ void InspectorListModel::buildModelsForEmptySelection()
         removeUnusedModels({}, false /*isRangeSelection*/);
         return;
     }
-
+
     removeUnusedModels({}, false /*isRangeSelection*/, persistentSectionList);

     createModelsBySectionType(persistentSectionList);
```

It basically is a tab or a couple spaces too much

👍 1

Add more commits by pushing to the **properties-no-score** branch on **AnnikaLevesque/MuseScore**.

All checks have passed
11 successful checks

Show all checks

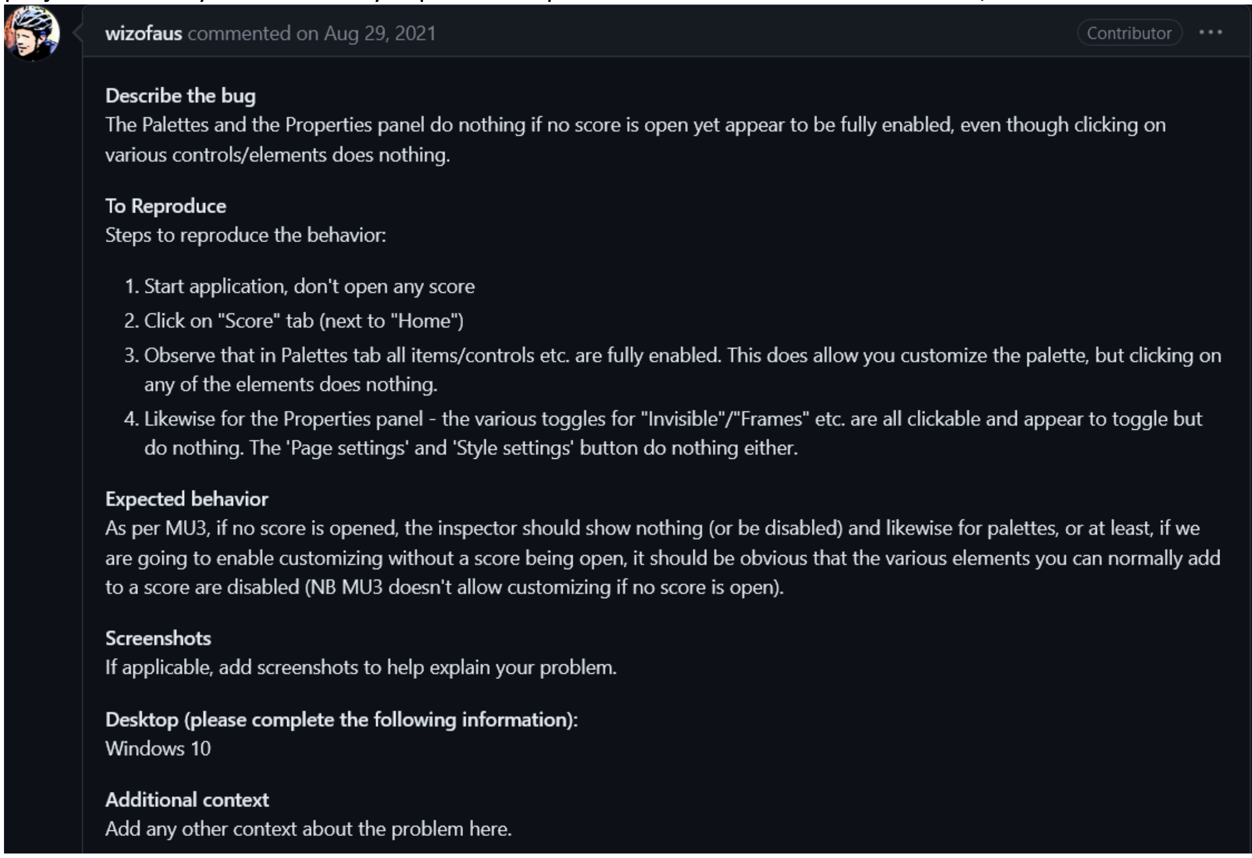
This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

9. **Plan updates.** I originally intended to make the panel completely disappear when no score was open, but upon further inspection of the program layout, I realized that wasn't the best approach; one guiding philosophy of MuseScore is to allow the user to customize the UI layout as much as possible, so panels can be dragged and dropped in different locations and many buttons can be shown or hidden at will. A project maintainer had mentioned that the palettes panel should remain even when there is no score to allow the user to customize it, and it is for this reason that I decided the properties panel should also remain; the user should be able to drag and drop it as needed. However, the buttons that didn't have any function could be removed. That being said, I would have liked to disable the buttons (see section 5 with the task description) instead of remove them entirely, but I could not figure out the code to make that happen, so I settled for having the modules disappear instead. The amount of time and effort each sub-task took in the timeline I proposed in HW6a was mostly accurate, except I over-estimated the time it would take to learn the project code style guidelines because there were few that pertained to my task. The biggest deviation was that instead of spreading the sub-tasks throughout April, I ended up having to complete all of it over the course of 2 days (April 18 and 19), during which I spent about 3 hours figuring out administrative details like how to compile and run changes, how to submit a pull request, etc., 6 hours reading and learning how the code was structured (in particular, the panel used to be named "inspector" instead of "properties", so it took me a while to learn I had to look for "inspector" in the source code to modify the "properties" panel), and about 4 hours implementing the change, testing it, and trying different options (e.g. removing the buttons one by one vs as a whole module). The process was relatively straightforward, likely because of my familiarity with the software and the clear requirements due to the previous attempt at fixing the issue (as mentioned in HW6a), but the timeline was severely truncated due to time conflicts with other classes, which was an unanticipated risk. I did not expect the GEO strike to impact my classes as heavily as it did, since most of my classes do not have GSIs.
10. **Your experiences and recommendations:** Overall I feel a great sense of accomplishment at being able to contribute to a program that has helped me out tremendously in the past and has come a long way since I last used it. It was really intimidating to work on a large code base that I was not familiar with, but since the Discord server was very active, I was able to get very quick support on things like compiling, finding relevant sections in the source code, etc. If it weren't for that quick communication, I would've had a lot more trouble (for example, the comment that a project maintainer gave that explained why I was failing the code style check). I agree with the recommendation that students choose a project that is very active.
    - Even with all the support I received, it was very difficult to figure out how to get all the dependencies I needed, compile and run the code, then compile and run changes, etc. The instructions for doing so were only partially complete, with an out of date version called the "developer's handbook" (https://musescore.org/en/handbook/developers-handbook) and an incomplete up to date version called the "contribution wiki" (https://github.com/musescore/MuseScore/wiki/Set-up-developer-environment). For example, I first tried to use Qt Creator as my IDE as suggested by the developer's handbook, but only after wasting a few hours trying to make it work, I found out in the Discord that Qt Creator is no longer compatible with the current version of MuseScore, and I should switch to Visual Studio instead for Windows development. Then when I followed the Visual Studio instructions, it said that I should only build the *INSTALL* project (one of the Visual Studio projects that came with the source code) once, but after asking around again I found out that it needs to be rebuilt with every modification to the source code. There was also a whole mess with installing packages in various

terminals that thankfully only needed to happen once because I still don't understand how it works.

- On the bright side, the fact that such issues are part of the previous homeworks assigned in EECS481 (i.e. getting a program to compile and work in a given developer environment) did help me get over those hurdles. In class we mentioned how documentation is an important but often overlooked part of software development, and I felt that keenly as I tried to piece together instructions from various sources just to get the program running, which presented a huge barrier to entry for new open-source contributors. I know it's not in the spirit of the homework, but I felt like what this project in particular needed help with was not necessarily programming, but documentation, translation, and many other tangential jobs that would greatly speed up development. Maybe in the future there could be some assignment asking students to help with these tasks.
- I was very surprised by how promptly members of the community responded to my questions both in the Discord server and on GitHub. I knew that the community was active, but since I had intentionally chosen an issue that wasn't too high on the priority list (to prevent it from getting sniped by another developer), I didn't quite expect to get such prompt responses while I was working on it. There were also responses directly from community leaders like Jojo-Schmitz (see screenshots in previous sections), who has been active in development since MuseScore3 at the latest (the current release is MuseScore4). I felt really welcome and supported by the community, and it makes me glad I chose this project to do this homework on.
- I also think it helped that I chose a project that had recently gone through a major overhaul to clean up a lot of technical debt. Since MuseScore was an open-source project to begin with, it had accumulated a lot of small, often unconnected contributions over the years, and when Ultimate Guitar acquired the project and released MuseScore4, a ton of the changes were just code refactoring, which made the code base really approachable to new contributors. Now that MuseScore has full-time paid project maintainers, it seems like the technical debt won't ever get as bad as it used to (although judging by a lot of the complaints on Discord, there are still some performance issues carried over from MuseScore3), which really shows the importance of some kind of central vision and leadership in open-source projects. This may not apply to every project, but I think it can often be helpful to have people working on a project that understand the code base more deeply than the average community contributor, and can lead (or even implement themselves) the kind of large-scale revisions/refactoring that are sometimes needed.
- In terms of task chosen, I'm glad I chose one that was relatively quick and easy to implement, since as I suspected, the vast majority of the time was spent familiarizing myself with the code base and figuring out how to implement any kind of change at all. That being said, I do think it was difficult to find evidence of QA for the homework assignment because my task wasn't one that lended itself to subtle bugs that could be teased out. Either the buttons appeared or they didn't. I did find one edge case where the user closed the score while not selecting anything (as described in previous sections), but that was about the extent of the information I learned from QA. Since that is such a large part of the class, it made it difficult to show my understanding of the material. If I were to do it again, I would perhaps choose another task that helps me show that more clearly.

- I also struggled to find ways to show requirements elicitation. One thing I think this project does very well is be very explicit and specific with what each issue is about;



wizofaus commented on Aug 29, 2021 Contributor

Describe the bug
The Palettes and the Properties panel do nothing if no score is open yet appear to be fully enabled, even though clicking on various controls/elements does nothing.

To Reproduce
Steps to reproduce the behavior:

1. Start application, don't open any score
2. Click on "Score" tab (next to "Home")
3. Observe that in Palettes tab all items/controls etc. are fully enabled. This does allow you customize the palette, but clicking on any of the elements does nothing.
4. Likewise for the Properties panel - the various toggles for "Invisible"/"Frames" etc. are all clickable and appear to toggle but do nothing. The 'Page settings' and 'Style settings' button do nothing either.

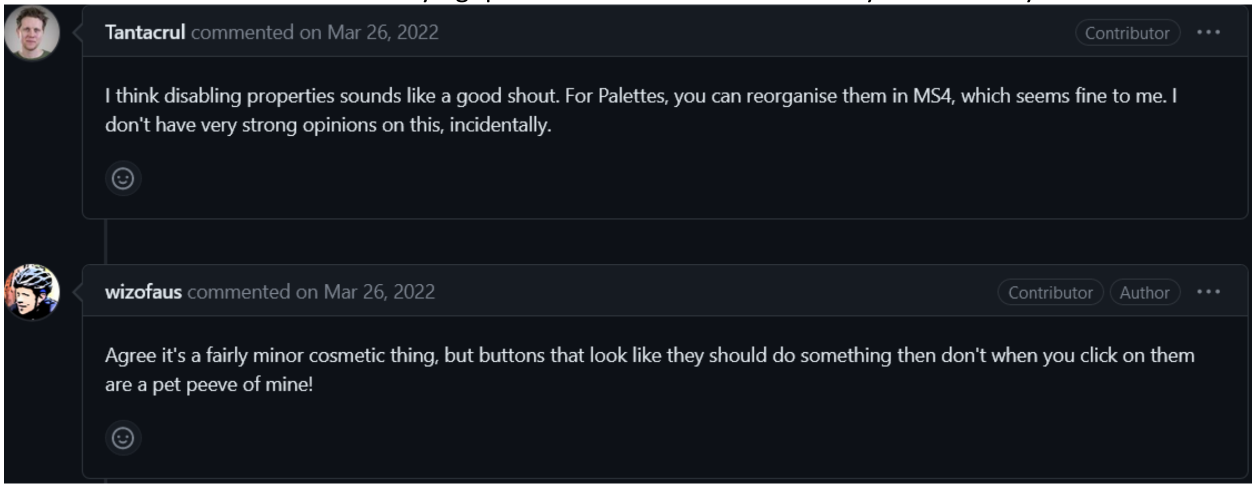
Expected behavior
As per MU3, if no score is opened, the inspector should show nothing (or be disabled) and likewise for palettes, or at least, if we are going to enable customizing without a score being open, it should be obvious that the various elements you can normally add to a score are disabled (NB MU3 doesn't allow customizing if no score is open).

Screenshots
If applicable, add screenshots to help explain your problem.

Desktop (please complete the following information):
Windows 10

Additional context
Add any other context about the problem here.

in the original issue submission, the description of the problem was very clear, as was the desired behavior (perhaps because it was submitted by a contributor, someone who understands MuseScore deeply and can describe things in a way easily understood by other contributors). This was a tremendous help in me understanding what the task was, and I didn't feel a need to ask clarifying questions because it seemed very clear already.



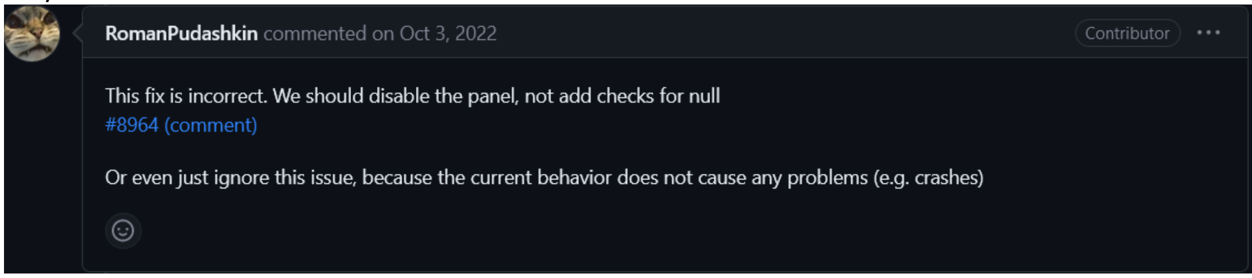
Tantacrul commented on Mar 26, 2022 Contributor

I think disabling properties sounds like a good shout. For Palettes, you can reorganise them in MS4, which seems fine to me. I don't have very strong opinions on this, incidentally.

wizofaus commented on Mar 26, 2022 Contributor Author

Agree it's a fairly minor cosmetic thing, but buttons that look like they should do something then don't when you click on them are a pet peeve of mine!

similarly, the comments on the issue further clarified the issue in terms of what the root of the problem was and the correct approach to solving it. Also, I was fortunate enough to have someone else already attempt (and fail) to solve the problem, so I had an example of what not to do as reference. The comments on that pull request were also very informative:


RomanPudashkin commented on Oct 3, 2022

This fix is incorrect. We should disable the panel, not add checks for null
#8964 (comment)

Or even just ignore this issue, because the current behavior does not cause any problems (e.g. crashes)

so in the end, I didn't see a way to prove my understanding of requirements elicitation, because any questions I came up with would just be wasting the project maintainers' time. I'm sure that if I spent more time looking, I could find a less well-defined issue to work on, but almost all the issues in the MuseScore project were very well formatted and when they weren't, the project maintainers themselves were quick to ask and comment on the GitHub page (often within 24 hours). I think this is a result of having very dedicated developers and a community that may be more technically savvy than the average userbase. As mentioned previously, MuseScore is the first free music notation software that truly has all the functionalities any composer would need, so many composers and musicians are used to using their own programming/technical skills to make the other (less functional) alternatives work. Even when I used MuseScore, before I learned any programming or computer science, I had to learn on YouTube and forums how to make workarounds for some of the things I wanted to include in my sheet music, and I think that spirit has lingered even as MuseScore becomes a more user-friendly program. Even so, for the sake of completing this homework, that did make the report more difficult to write.

- I really think that MuseScore is one of the easiest open-source projects to contribute to, and I would really recommend it for students in future semesters (not only because I am fond of the project and want more people to help it grow). There are plenty of tasks (such as the one I did for this homework) that don't require any background in music or composition, although such a background would probably help students trying to contribute to MuseScore. MuseScore is even one of the projects that has been selected for Google's Summer of Code (https://summerofcode.withgoogle.com/programs/2023/organizations/musescore) starting 2022, which is a good opportunity for students and beginner developers.

11. **Advice for future students:** One thing that really helped me in this class was having a Discord server (or some other kind of group communication) that would ping students about upcoming due dates, especially the pop quizzes that appear on Gradescope.
12. I am willing to let future students see my materials.

Appendix - Running test results:

```
Rebuild started...
1>-- Rebuild All started: Project: ZERO_CHECK, Configuration: RelWithDebInfo x64 --
1>Checking Build System
2>-- Rebuild All started: Project: RUN_TESTS, Configuration: RelWithDebInfo x64 --
2>Test project C:/Users/jingy/MuseScore/msvc.build_x64
2>      Start  1: global_tests
2> 1/17 Test  #1: global_tests .....................***Failed    0.78 sec
2>      Start  2: accessibility_tests
2> 2/17 Test  #2: accessibility_tests .............   Passed    0.17 sec
```

```
2>      Start  3: audio_test
2> 3/17 Test  #3: audio_test .......................Exit code 0xc0000135
2>***Exception:   0.01 sec
2>      Start  4: draw_tests
2> 4/17 Test  #4: draw_tests .......................   Passed    0.17 sec
2>      Start  5: mpe_test
2> 5/17 Test  #5: mpe_test .........................   Passed    0.18 sec
2>      Start  6: ui_tests
2> 6/17 Test  #6: ui_tests .........................   Passed    0.44 sec
2>      Start  7: diagnostics_tests
2> 7/17 Test  #7: diagnostics_tests ................   Passed    1.33 sec
2>      Start  8: engraving_tests
2> 8/17 Test  #8: engraving_tests ..................***Failed  248.83 sec
2>      Start  9: iex_bb_tests
2> 9/17 Test  #9: iex_bb_tests .....................   Passed    0.67 sec
2>      Start 10: iex_braille_tests
2>10/17 Test #10: iex_braille_tests ................***Failed    2.87 sec
2>      Start 11: iex_bww_tests
2>11/17 Test #11: iex_bww_tests ....................   Passed    0.65 sec
2>      Start 12: iex_capella_tests
2>12/17 Test #12: iex_capella_tests ................***Failed   10.03 sec
2>      Start 13: iex_midi_tests
2>13/17 Test #13: iex_midi_tests ...................   Passed    0.62 sec
2>      Start 14: iex_musicxml_tests
2>14/17 Test #14: iex_musicxml_tests ...............***Failed   23.98 sec
2>      Start 15: iex_guitarpro_tests
2>15/17 Test #15: iex_guitarpro_tests ..............***Failed   54.69 sec
2>      Start 16: plugins_tests
2>16/17 Test #16: plugins_tests ....................   Passed    7.73 sec
2>      Start 17: project_test
2>17/17 Test #17: project_test .....................   Passed    0.26 sec
2>
2>59% tests passed, 7 tests failed out of 17
2>
2>Total Test time (real) = 353.52 sec
2>
2>The following tests FAILED:
2>	  1 - global_tests (Failed)
2>	  3 - audio_test (Exit code 0xc0000135)
2>	  8 - engraving_tests (Failed)
2>	 10 - iex_braille_tests (Failed)
2>	 12 - iex_capella_tests (Failed)
2>	 14 - iex_musicxml_tests (Failed)
2>	 15 - iex_guitarpro_tests (Failed)
2>Errors while running CTest
2>Output from these tests are in:
C:/Users/jingy/MuseScore/msvc.build_x64/Testing/Temporary/LastTest.log
2>Use "--rerun-failed --output-on-failure" to re-run the failed cases verbosely.
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: The command "setlocal
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: "C:\Program Files\Microsoft Visual
Studio\2022\Community\Common7\IDE\CommonExtensions\Microsoft\CMake\CMake\bin\ctest.e
xe" --force-new-ctest-process -C RelWithDebInfo
```

```
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: if %errorlevel% neq 0 goto :cmEnd
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: :cmEnd
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: endlocal & call :cmErrorLevel %errorlevel% & goto :cmDone
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: :cmErrorLevel
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: exit /b %1
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: :cmDone
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: if %errorlevel% neq 0 goto :VCEnd
2>C:\Program Files\Microsoft Visual
Studio\2022\Community\MSBuild\Microsoft\VC\v170\Microsoft.CppCommon.targets(159,5):
error MSB3073: :VCEnd" exited with code 8.
2>Done building project "RUN_TESTS.vcxproj" -- FAILED.
========== Rebuild All: 1 succeeded, 1 failed, 0 skipped ==========
========== Rebuild started at 8:43 PM and took 05:54.594 minutes ==========
```