

EECS 481 Homework 6b

Albert Morgese
amorgese@umich.edu

April 16, 2018

Selected Project

For this assignment, I have contributed to the OpenRCT2 project (<https://github.com/OpenRCT2/OpenRCT2>). OpenRCT2 is an open-source reverse-engineering of Chris Sawyer's Roller Coaster Tycoon 2, reimplemented in C++. The reimplementation has been complete since December 2016, but many new features have since been added, as well as fixes for a number of bugs from the original game.

Project Context

OpenRCT2, which began in April 2014, follows in the tradition of OpenTTD, a similar reverse-engineering project from about ten years prior of Chris Sawyer's previous game, Transport Tycoon Deluxe. Like OpenTTD, OpenRCT2 was created out of a desire to extend the original game's features and support additional modern platforms. Some of the notable added features include online multiplayer, arbitrary resolutions, OSX/Linux/Android support, fast-forwarding, and hardware rendering via OpenGL.

One important detail about OpenRCT2 is that it still requires a copy of the original game to import assets from, even though all the code of the game has been reimplemented. This contrasts with OpenTTD, whose community has created free asset packs from scratch for distribution with the game. As such, OpenRCT2 does not have the immediate accessibility of OpenTTD, but it is still very much the preferred way to play Rollercoaster Tycoon and Rollercoaster Tycoon 2 on a modern computer.

Project Governance

OpenRCT2 has fairly well-structured contribution protocols. The project makes use of the "gitflow" workflow, which essentially involves a master branch for stable releases, a development branch for the latest changes, and individual feature and hotfix branches as necessary. They have some standards for commit messages (<https://github.com/OpenRCT2/OpenRCT2/wiki/Commit-Messages>), but these seem to only be loosely followed.

OpenRCT2 has a small core development team with push access to the repository. Features and bug fixes are generally added by GitHub pull requests (even from team members), and the changes are reviewed by at least one team member, including checking for compliance with the coding style guide (<https://github.com/OpenRCT2/OpenRCT2/wiki/Coding-Style>). If approved, a team member accepts the pull request. Additionally, continuous integration builds run for each pull request, to ensure that the game builds properly for every platform. If the change involves a modification to user-facing text, a new issue is created in the separate localization repository, where contributors can add translations for a number of languages.

There is also a live chatroom for discussion among contributors, hosted on gitter (<https://gitter.im/OpenRCT2/OpenRCT2>). The channel has a fairly low level of activity, but most team members are present, and responses usually come quickly.

Since OpenRCT2 is a game, automated testing is difficult for many parts of the codebase. There do exist a handful of unit tests in the project, but these are a relatively recent addition, and they have not yet been added to the continuous integration system. The majority of quality assurance is done via code review and manual testing.

Task 1: Multiplayer window should have different titles for each tab

Description

This is a small, low-priority task, which I chose specifically due to having the label “good first issue”. In essence, there is an (in-engine) window with a tab for each multiplayer-related feature, which previously had the title “Multiplayer” for all tabs. The task was to change this window to display a different title for each tab, to be more consistent with other tabbed windows in the game.

This issue sounds nearly trivial, and it was indeed quite easy to implement. However, it did involve a meaningful amount of code comprehension, and it also required interacting with the project’s localization system. The windowing system in OpenRCT2 is fairly daunting at first sight, involving large arrays of inscrutable numerical constants and repeated references to the same widgets in multiple places with unclear significance. Despite this, progress went fairly smoothly, as a simple matter of understanding how the title was previously stored; understanding how other windows with changing titles implemented this feature; and implementing changing titles for this window in the same manner.

Submitted Artifacts

<https://github.com/OpenRCT2/OpenRCT2/pull/7363>

This is a link to the pull request for this task.

Task 2: “Uppercase banners” option should properly handle non-ASCII characters

Description

This task initially appeared small, but ended up taking the most effort out of the three. There is an option in the game to automatically uppercase text on banners, but it previously only uppercased ASCII characters, leaving non-ASCII lowercase (UTF-8) characters untouched. The bug report contained discussion about the underlying cause of the issue (using the C standard library’s `toupper` function), but no attempts were made to fix the bug.

I originally assumed that fixing this bug would be as simple as finding the appropriate C or C++ standard library function for handling unicode uppercasing, and replacing it *in situ*. However, I quickly realized that neither of these standard libraries have any meaningful support for unicode operations. Additionally, I discovered that OpenRCT2 did not have any external library dependencies for unicode handling, and instead implemented its own conversions and operations as needed. As such, fixing this issue required reading through the unicode specification, scraping codepoint data, putting the conversions into a large table, and writing a large conversion function to handle all possible UTF-8 byte lengths.

After submitting this, several members of the core development team commented that it would be preferred to solve this by integrating the project with an external library for handling unicode (namely ICU). The owner of the project then requested that I attempt this task for Linux, as well as removing the other existing conversion tables. I was unable to integrate the library with the build system properly, but I did succeed at replacing the various conversion functions with calls to ICU. At present, these changes are functional, but they will need additional work and verification to update the build system properly and ensure that there are no multiplatform issues introduced.

Submitted Artifacts

<https://github.com/OpenRCT2/OpenRCT2/pull/7385>

This is a link to the pull request for this task.

Task 3: Add “rotate to face path” option to shops

Description

In contrast to the previous tasks, this task was based off of a feature request, rather than a bug report. In OpenRCT2, shops are 1-tile entities with an entrance on (usually) a single side, and (prior to this change) must be manually rotated into the correct orientation so guests can access the entrance. However, when placing shops next to existing footpaths, it is very rare to want the shop to face any direction other than towards the path. As such, this new feature would add auto-rotation to save the player unnecessary effort.

Implementing this feature was reasonably straightforward; most of the effort went towards understanding the ride construction code and finding the appropriate location along the pipeline to implement the feature. The initial implementation also involved working again with the windowing system to add a checkbox option for the feature. However, later community discussion on the pull request resulted in consensus that the feature should be non-optional, so I reverted this part of the change.

The basic implementation consists of a check for a path at each tile adjacent to the shop, and if a path exists, the shop is rotated to face it. Following discussion in the pull request, I added several additional improvements to this, including only rotating the shop if it is not already facing a path, and not rotating to face an inaccessible path (in the case of sloped paths). Since all “ghost” rendering is done client-side, no changes were required to preserve multiplayer behavior.

Submitted Artifacts

<https://github.com/OpenRCT2/OpenRCT2/pull/7411>

This is a link to the pull request for this task.

QA Strategy

OpenRCT2’s quality assurance is dominated by manual testing and pass-around code review. As such, the majority of my QA effort was in these two categories. All three of these tasks were fairly easy to test manually, as they could all be tested within seconds of starting a new game. One interesting detail of the manual testing process is that it was particularly important to test the changes in multiplayer mode with an unmodified server; if any of the changes failed this test, it would mean that an increment to the network version would be required, to ensure that clients do not connect to incompatible servers.

The project has fairly few formal guidelines for code review, so participating in the code review process was simply a matter of submitting a pull request and waiting for comments. As expected, the majority of the comments I received dealt with non-functional code improvements (cf. Bacchelli and Bird 2013), such as adhering to the style guide, using named constants over magic numbers, and using helper functions over accessing bitflags directly.

Task 2, unlike the other tasks, presented a good opportunity for unit testing. Although there were fairly few existing unit test files, there did exist a test suite for various string utility functions, so I added several tests to ensure proper functionality. These tests also made the development process significantly easier, as they allowed for easily viewing the precise output of the function, rather than trying to view the rendered output in-game.

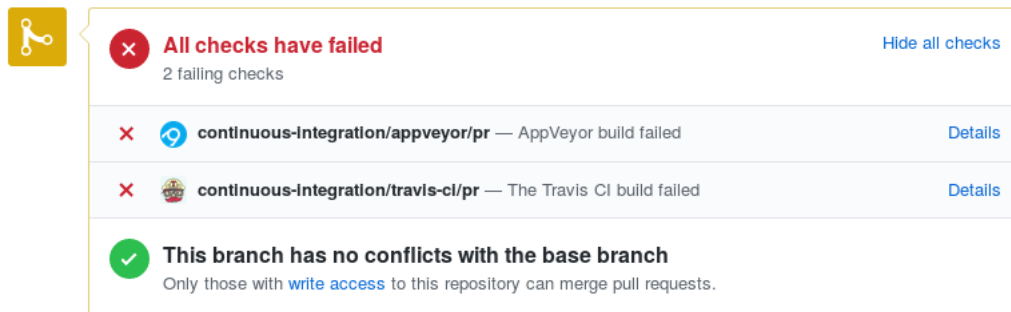
QA Evidence

Each task pull request contains code review comments and continuous integration build checks.



```
11 test/tests/StringTest.cpp
-64,3 +64,14 TEST_F(StringTest, Split_ByEmpty)
64 {
65     EXPECT_THROW(String::Split("string", ""), std::invalid_argument);
66 }
67 +
68 +TEST_F(StringTest, ToUpper_Basic)
69 +{
70 +     auto actual = String::ToUpper("test TEST tEsT 1234");
71 +     ASSERT_STREQ(actual.c_str(), "TEST TEST TEST 1234");
72 +}
73 +TEST_F(StringTest, ToUpper_Unicode)
74 +{
75 +     auto actual = String::ToUpper("éèóèðððððð éèóèðððððð");
76 +     ASSERT_STREQ(actual.c_str(), "ÉÈÓÈÐÐÐÐÐÐ ÉÈÓÈÐÐÐÐÐÐ");
77 +}
```

Figure 1: Source code for tests added in task 2 (<https://github.com/OpenRCT2/OpenRCT2/pull/7385/files#diff-64bf2fdfca6b9247b05ca6948a8ad276>)



All checks have failed Hide all checks
2 failing checks

- continuous-integration/appveyor/pr** — AppVeyor build failed Details
- continuous-integration/travis-ci/pr** — The Travis CI build failed Details

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

Figure 2: Failed CI build results from task 2 (which has an invalid CMake configuration)



Broxzier requested changes 2 days ago

[View changes](#)

Nicely done, this works exactly like how I envisioned it! There are only a few things in the code that I'd like to see adjusted before this can get merged.

```
src/openrct2-ui/windows/RideConstruction.cpp Hide outdated  
3433 + ride_type_has_flag(ride->type, RIDE_TYPE_FLAG_IS_SHOP)  
3434 + {  
3435 + rct_tile_element *pathsByDir[4];  
3436 + constexpr std::pair<uint8, uint8> DirOffsets[4] = {{-1, 0}, {0, 1}, {
```



Broxzier 2 days ago • edited Member

Instead of `std::pair<...>` use `sLocationXY8`.



Gymnasiast 2 days ago Member

No, that's improper use of `LocationXY8`, which takes Tile Coordinates. `std::pair` is fine.



Broxzier 2 days ago Member

Yes, but this is `sLocationXY8` (with the `s` prefix), which used signed bytes and can't store the coordinates for every possible tile. `DirOffsets[i].x` and `.y` rather than `DirOffsets[i].first` and `.second` is also clearer in my opinion.



Gymnasiast 2 days ago Member

Right, ok. Let's go with your suggestion then.



Reply...

```
src/openrct2-ui/windows/RideConstruction.cpp Show outdated
```

```
src/openrct2-ui/windows/RideConstruction.cpp Show outdated
```

```
src/openrct2-ui/windows/RideConstruction.cpp Show outdated
```

Figure 3: An example code review comment from task 3

Plan Updates

Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.

– Douglas Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid*

Task/subtask	Time	Cumulative
Task 1		
Understanding relevant code	0:20	0:20
Planning and implementing change	0:10	0:30
Verifying and submitting change	0:20	0:50
Code review revisions	0:10	1:00
Task 2		
Understanding relevant code	0:50	1:50
Planning and implementing change	3:00	4:50
Verifying and submitting change	0:50	5:40
Code review revisions	4:00	9:40
Task 3		
Understanding relevant code	1:00	10:40
Planning and implementing change	1:30	12:10
Verifying and submitting change	0:50	13:00
Code review revisions	2:10	15:10

Table 1: Retrospective time schedule

As I somewhat suspected, my effort estimation was far too optimistic. Thankfully, I selected tasks with this possibility in mind, so adjusting my plan to compensate was fairly simple, and I was able to simply drop the fourth and fifth tasks I planned to complete. Task 1 was already essentially complete by the time I wrote the initial time plan, so there was little possibility for error in the “estimates” for this task.

For task 2, however, I vastly underestimated the scope of changes required. In particular, I assumed the existence of some easily accessible function for performing the uppercasing task, but since such a function did not already exist, I spent a considerable amount of time implementing my own. After finishing this implementation, the result of code review was that I should instead integrate an external library for the functionality instead, and additionally convert several other parts of the codebase to use this library. As such, I spent a great deal of additional time essentially starting from scratch.

Task 3 was closer to my estimate, but still took considerably longer than expected, specifically in the review process. The initial changes only took slightly longer than I imagined, but a number of requests were made for changes in the behavior of the feature (as well as a handful of code quality changes), which I did not adequately prepare for in my estimate.

Experiences and Recommendations

Overall, I had a very positive experience working with OpenRCT2. The team members were very polite, and they all clearly put a great deal of effort into making the project succeed. The quality of the codebase also surpassed my expectations; although there are certainly many visible remnants of the reverse-engineering history of the project, a great deal of the codebase has been refactored extensively to resemble any other typical C++ project. Although the codebase is reasonably large, it is significantly smaller than codebases I have worked on in industry (including a C++ game engine), so the size of the codebase did not significantly hinder my contributions.

Communication

I was particularly impressed by the speed and quality of comments on changes from the development team. One comment on the pull request for task 3 even included animated screen captures of issues with the feature's behavior, along with detailed descriptions. The effort present was nearly as much as I've seen in code review processes in industry, which is surprising for entirely volunteer work. The developer chatroom was also a very valuable resource, and questions I had regarding project setup or code style were always answered reasonably quickly.

When I was initially deciding which project to contribute to for this assignment, one feature of OpenRCT2 that drew me to the project, aside from the generally good level of activity, was the presence of a "good first issue" tag on a number of issues in the tracker. Although the tag only seemed to be occasionally applied, it was very reassuring to work on a task explicitly marked for newcomers to the project. Indeed, the two tasks I worked on with this tag were fairly straightforward, while the task I worked on without this tag (task 2) was far more complicated than it initially appeared to be.

Codebase

One interesting property of the codebase is that despite having a detailed code style guide, large portions of the codebase do not adhere to the style guide (e.g. functions should be title case, but are instead snake case). It appears that the style guide was created after most or all of the original reverse-engineering was completed, as most of these functions also reference addresses in the original executable. (In addition to these nonconforming sections, there are also a number of functions that have yet to receive descriptive names, and instead have names such as `sub_98196C`, which makes comprehension very difficult.)

This is an issue I have also occasionally encountered in industry. It tends to make for a frustrating coding experience, as much of my strategy when implementing new features in an unfamiliar codebase is to simply find other segments of the codebase implementing similar tasks, and basing my changes off of that. In particular, this was my strategy for task 1, but in code review, it was pointed out that my changes did not actually conform to the style guide, as the code I based them off of originally also did not conform to the style guide. There may exist automated refactoring tools that could help solve this immediate issue, but it appears that there is still also a great deal of more nuanced refactoring left to do within these nonconforming functions (e.g. improving variable names and code layout).

The largest stumbling block to contributing effectively to this project was dealing with the CMake build system. This was my first experience attempting to understand and modify a CMake configuration, and when I was requested to add an external library dependency for task 2, my efforts were so fruitless that I eventually cut my losses and gave up. Thankfully, one of the team members then offered to handle the CMake changes, so I was able to continue work on the task. However, this experience has left me with a very poor impression of CMake; the documentation is limited, the script semantics are awkward, and the error messages are inscrutable.

Things I Would Do Differently

OpenRCT2 is a well-organized project, and I have few issues to take with it. In particular, the git workflow works well with the project's organization, the localization system is very effective for avoiding cluttering the main repository, and the development team does a very good job of ensuring quality through code review. However, there are still a handful of places where I would have made significantly different decisions if I were starting a new project from scratch.

The project in general seems to be lacking a certain amount of direction. There are plenty of open feature requests and bug reports, and development is reasonably active, but it is very unclear what the high-level goals are for the project, if any (e.g. refactor all the original reverse-engineered code, fix all desynchronization issues in multiplayer). For the level of contribution I made in this assignment, this was not a salient problem,

but if I were looking to attract long-term contributors to a project of my own, I would try to have at least a more formal prioritization of high-level “epic” issues, if not an explicit “plan” for the future of the project.

Additionally, as previously mentioned, inconsistency of existing code with a project’s style guide is very frustrating. I am of the opinion that the main value of style guides is enforcing consistency, regardless of what the particular guidelines are. As such, if I were to find myself in OpenRCT2’s position, I would not attempt to change the coding style mid-project, unless I was willing to undertake the refactoring effort as a high-priority task.

In terms of the codebase itself, the existing test suites are rather neglected. Though certainly better than no tests at all, they seem to be generally ignored by most contributors most of the time. One of them even remains broken, despite having only a handful of test suites. In a project of my own design, I would attempt to be more formal about testing standards, and I would particularly prioritize integrating the tests with the continuous integration system to make test results much more prominent.

Advice for Future Students

“If you enjoy the sections of the class involving automated techniques for static and dynamic analysis, I highly recommend taking EECS 590 with Wes as well.”

I am happy to let future students see my work, attributed or anonymized.

Evidence of Accepted Changes

- Park Joon-kyu (segjaun07) - Misc.
- Harrison Gentry (hgentry) - Date-changing command, misc.
- Joshua Moerman (Jaxan) - Misc.
- Nicolas Hawrysh (xp4xbox) - Misc.
- Albert Morgese (Fusxfaranto) - Misc.

Bug fixes

- (halfbro)
- (Myrtle)
- (noon)

Figure 4: A snippet of the current `contributors.md` file in OpenRCT2 (<https://github.com/OpenRCT2/OpenRCT2/blob/develop/contributors.md>)

Task 1’s merged pull request is visible at <https://github.com/OpenRCT2/OpenRCT2/pull/7363>.

Task 3’s merged pull request is visible at <https://github.com/OpenRCT2/OpenRCT2/pull/7411>.