Optimizing the SIGBOVIK 2018 speedrun

leo60228
@leo60228

April 1, 2020

Abstract

SIGBOVIK 2018 [1] is a popular game. However, very little work has been put into optimizing its speedrun. In this paper, I will demonstrate usage of pathfinding algorithms for the goal of making the optimal speedrun of SIGBOVIK.

1 Motivation

I was bored.

2 Method

I first looked at implementations of pathfinding algorithms in Rust. I initially wanted to use A* because it was popular. However, the API looked complicated. I found an implementation of DFS to use instead. This DFS implementation required me to define a start point, one or more end points, and a function to map from point to points. I implemented all three of these, using an enum of Start, Success, and Page, because I forgot that there was only one success page.

I used a depth first search algorithm in order to create the mapping. To do this, I clicked as far as possible along the first choices, adding to my mapping (a `match` block) as I went, then backtracking once I reached a cycle or the end. I then manually removed cycles because I wasn't sure if the DFS implementation I was using supported them and I didn't feel like writing my own.

After I had defined these three functions, I simply used the `pathfinding` [2] crate's DFS function. This gave me an optimal path to program into LiveSplit One [3], a tool for speedrunning that I've seen speedrunners use, so I probably should too if I'm speedrunning. I opened it on my phone and added the order that my code gave me.

I recorded my phone and computer screen while I began the speedrun. I then combined the phone and computer screen and attempted to sync them from memory in a video editor. I then uploaded it to YouTube [4].

3 Results

My speedrun was 17.88 seconds, which I'm pretty sure is a world record among the SIGBOVIK 2018 speedrunning community. It can be viewed at `https://youtu.be/VPrT8Y-aRRs`. My code will be available on GitHub at `https://github.com/leo60228/sigbovik2018` if I remember to make it non-private after SIGBOVIK.

4 Appendix: Code

```
use pathfinding::directed::dfs::dfs;

#[derive(Debug, PartialEq, Eq, Copy, Clone)]
pub enum Page {
    Start,
    Success,
    Page(isize),
}

macro_rules! pagevec {
    ($($page:expr),*) => {
        vec![$(Page::Page($page)),*]
    }
}

impl Page {
    pub fn successors(&self) -> Vec<Self> {
        match self {
            Page::Start => pagevec![47, 177, 205],
            Page::Page(47) => pagevec![153, 206],
            Page::Page(153) => pagevec![17],
            Page::Page(17) => pagevec![35, 135],
            Page::Page(35) => pagevec![51, 68],
            Page::Page(51) => pagevec![68], // cycle
            Page::Page(68) => vec![], // cycle
            Page::Page(135) => pagevec![124, 183],
            Page::Page(124) => pagevec![116, 88],
            Page::Page(116) => pagevec![208],
            Page::Page(208) => vec![Page::Success],
            Page::Page(88) => pagevec![208],
            Page::Page(183) => pagevec![207, 208],
            Page::Page(207) => vec![],
            Page::Page(206) => vec![],
            Page::Page(177) => pagevec![130, 117],
            Page::Page(130) => pagevec![154, 39],
```

```
            Page::Page(154) => pagevec![17],
            Page::Page(39) => vec![],
            Page::Page(117) => pagevec![87, 50, 28],
            Page::Page(87) => pagevec![69, 40, 28],
            Page::Page(69) => vec![], // cycle
            Page::Page(28) => vec![], // cycle
            Page::Page(50) => vec![], // cycle
            Page::Page(40) => pagevec![178], // cycle
            Page::Page(178) => pagevec![208], // cycle
            Page::Page(205) => vec![],
            Page::Success => pagevec![],
            _ => unimplemented!("{:?}", self),
        }
    }
}

fn main() {
    println!("{:?}", dfs(Page::Start, Page::successors, |x| x == &Page::Success));
}
```

5 Appendix 2

I just realized I probably should have used more sections, so here's one.

References

[1] Association for Computational Heresy. Message from the Organizing Commitee. In *Proceedings of SIGBOVIK 2018*. ACH, Pittsburgh, PA, USA, March 29, 2018.

[2] Samuel Tardieu. `pathfinding`. `https://crates.io/crates/pathfinding`

[3] LiveSplit. LiveSplit One. `https://one.livesplit.org`

[4] Google. YouTube. `https://youtube.com`