Background

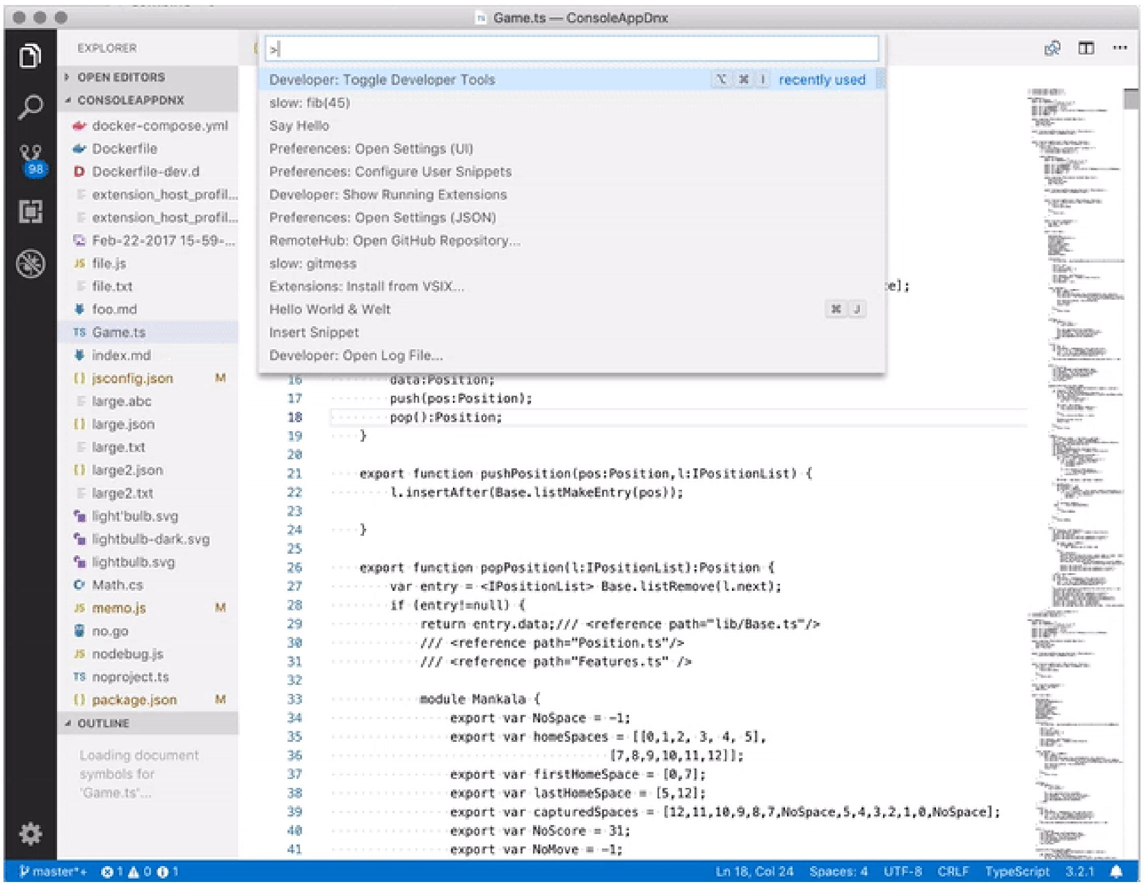
Visual Studio Code runs extensions in a separate Node.js process – the extension host. This isolates extension code from VS Code itself, but it doesn't isolate different extensions from each other. Due to the single threaded nature of JavaScript, an extension can, willingly or not, monopolize the extension host thread, block all other extensions, and make operations appear as unresponsive.

To detect this situation, VS Code monitors the extension host so that it knows when it becomes unresponsive. It then attaches a CPU profiler for a short while to analyze what part of the extension host has been causing the unresponsiveness.

Dealing with high CPU load issues

When an extension is monopolizing the extension host process, VS Code encourages users to file an issue against the extension in question. When you receive such an issue, you should do the following:

- Make sure the `<myExtensionId>.cpuprofile.txt` -file has been uploaded. If not, ask the author for it.
- Remove the `.txt` -suffix
- Open the JavaScript profiler via **F1** > **Developer Tools** > ⋮ > **More Tools** > **JavaScript Profiler**
- Select **Load** and pick the `.cpuprofile` -file

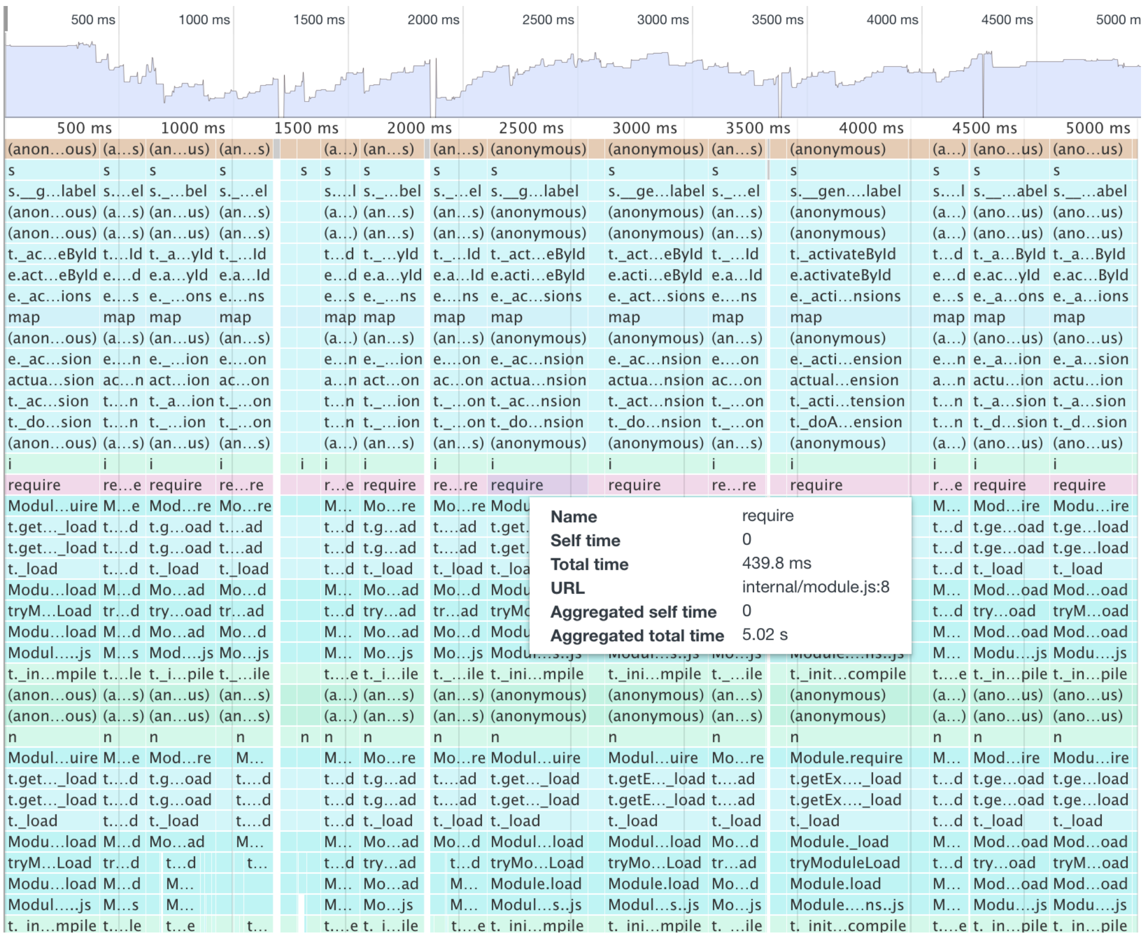


```
        data:Position;
        push(pos:Position);
        pop():Position;
    }

    export function pushPosition(pos:Position,l:IPositionList) {
        l.insertAfter(Base.listMakeEntry(pos));

    }

    export function popPosition(l:IPositionList):Position {
        var entry = <IPositionList> Base.listRemove(l.next);
        if (entry!=null) {
            return entry.data;/// <reference path="lib/Base.ts"/>
            /// <reference path="Position.ts"/>
            /// <reference path="Features.ts" />

            module Mankala {
                export var NoSpace = -1;
                export var homeSpaces = [[0,1,2, 3, 4, 5],
                                         [7,8,9,10,11,12]];
                export var firstHomeSpace = [0,7];
                export var lastHomeSpace = [5,12];
                export var capturedSpaces = [12,11,10,9,8,7,NoSpace,5,4,3,2,1,0,NoSpace];
                export var NoScore = 31;
                export var NoMove = -1;
```

Use the JavaScript profiler to drill into individual functions and analyse in what part of the code most time has been spend. Keep in mind that a huge file, a huge number of files, or some other "unexpected" input may have been involved and that was the cause of the high CPU load.

Code Loading

Sometimes it's not code execution but code loading (which is also blocking) which causes unresponsive extension host. Profile then are harder to read and often look like this:



Name	require
Self time	0
Total time	439.8 ms
URL	internal/module.js:8
Aggregated self time	0
Aggregated total time	5.02 s

The problem is that there are too many, often little, files that all need to be read and loaded. Considering bundling your extension, e.g with https://webpack.js.org or https://rollupjs.org/. For the former there is a sample: https://github.com/Microsoft/vscode-extension-samples/tree/master/webpack-sample

Testing Changes

If you can reproduce the slowdown or if you want to verify your changes, you can also profile the extension host yourself. Follow this guide: https://github.com/Microsoft/vscode/wiki/Performance-Issues#profile-the-running-extensions