# Tiley 测试文档

## 目录

# 一．测试环境：

### 虚拟机：

VirtualBox 虚拟机环境
CPU：Intel Core i7-10870H @ 2.20GHz（1 vCPU 分配）
内存：7.8 GiB（虚拟机分配），Swap 4 GiB
显卡：VirtualBox/VMware SVGA II Adapter

### 测试系统：

操作系统：Ubuntu 24.04.2 LTS (noble)
内核版本：Linux 6.14.0-27-generic (PREEMPT_DYNAMIC)
架构：x86_64
虚拟化：KVM / full virtualization

### 系统资源：

内存状态：已用 2.3 GiB / 可用 5.5 GiB
CPU 缓存：L1d 32 KiB, L1i 32 KiB, L2 256 KiB, L3 16 MiB

### 显卡与显示：

GPU：VMware SVGA II Adapter
显示环境：Wayland（XDG_SESSION_TYPE=wayland）

# 二．测试方法：

目前以手动测试验证功能为主（结果仅仅只是多次手动测试得到的结果仅供参考，后期会进行脚本覆盖测试），测试性能如帧率等，内存安全测试.，稳定性测试。

# 三．测试时间：

2025 8.14 -2025 8.17
测试人员：爱吃橘子的 doro

# 四．功能测试

### 功能点：新建窗口（手动测试）

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果 |
|---|---|---|---|---|
| 1 | 新建 1 个窗口 | 新建窗口（按 F1），打印输出容器树内是否添加了的新的节点 | 会显示出一个窗口 | 通过 |
| 2 | 新建 3 个窗口 | 按 3 次新建窗口，打印输出容器树内是否添加了的新的节点并且按照顺序排列 | 会新建三个窗口，并根据鼠标位置在对应容器内分割 | 通过 |
| 3 | 新建 20 个窗口 | 按 20 次 F1，打印输出容器树内是否添加了的新的节点并且按照顺序排列 | 会正常新建 20 个窗口（平铺），但明显后面算然正常分割，却看不清 | 通过 |
| 4 | 测试在窗口两侧新建会在窗口对应侧新建 | 分别把鼠标放在左右两侧，输出鼠标位置（ cursor 坐标），窗口位置，查看是否匹配 | 鼠标放左侧，会新建在左侧，在右侧就新建在右侧 | 通过 |
| 5 | 测试在窗口聚焦区域外新建窗口 | 把鼠标放在 tiley 窗口外，新建窗口，，输出鼠标位置（ cursor 坐标），窗口位置，查看是否匹配 | 按默认顺序进行容器的分割 | 通过 |

### 功能点：切换壁纸（手动测试）

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 1 | 正常切换一个壁纸 | 按 crtl+t,打开对应 jpg,png,jpeg 文件 | 壁纸正常切换 | 通过 |
| 2 | 打开其它格式的文件 | 按 crtl+t,打开非 jpg,png,jpeg 格式文件 | 无法点击打开，加载默认壁纸 | 通过 |
| 3 | 选择多个符合格式的文件 | 按住 crtl，选择多个符合格式的文件 | 无法点击打开，加载默认壁纸 | 通过 |
| 4 | 选择多个不符合格式的文件 | 选择多个不符合格式的文件（jpg,png,jpeg 之外的文件） | 无法点击打开，加载默认壁纸 | 通过 |
| 5 | 切换一个图片损坏的壁纸 | 选取损坏的图片进行打开 | 无法点击打开，加载默认壁纸 | 通过 |
| 6 | 加载一个巨大的图片 | 选取一个 1200X1200 的图片进行打开 | 有理论最大极限值，无法打开 | 通过 |

**功能点：自动平铺（手动测试）**

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果 |
|---|---|---|---|---|
| 1 | 新建窗口自动平铺 | 新建一个窗口，打印输出我们的容器树，查看其叶子节点与容器的关系是否符合，新建窗口为子节点，原窗口左移/右移，并存储在一个新的父节点（容器下） | 自动平铺 | 通过 |
| 2 | 新建多个窗口自动平铺 | 新建一个窗口，打印输出我们的容器树，查看其叶子节点与容器的关系是否符合，各个层 | 默认自动平铺 | 通过 |

| | | | | |
|---|---|---|---|---|
| | | 级。 | | |
| 3 | 鼠标聚焦自动平铺 | 在鼠标在的窗口新建窗口，并输出当前的聚焦的窗口，判断是否为目标窗口 | 在鼠标在的窗口进行容器分割平铺 | 通过 |
| 4 | 浮动窗口拖放后自动平铺 | 把浮动状态的窗口拖到目的容器里，打印输出容器树查看是否添加成功并正常排布 | 浮动状态的窗口回到平铺状态 | 通过 |
| 5 | | | | |

**功能点：快捷键热重载功能（手动+脚本测试）**

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果 |
|---|---|---|---|---|
| 1 | 切换快捷对应功能可实时切换 | 把 json 文件绑定的对应的函数名切换，并打印输出存储用的map里是否记录到里新的关系 | 程序运行时切换后，二者快捷键使用后功能交换了 | 通过 |
| 2 | 切换 10 个快捷键功能 | 把 json 文件里的快捷键与对应功能切换 10 个，并打印输出存储用的map里是否记录到里新的关系 | 程序运行时切换后，功能一一对应 json 文件 | 通过 |
| 3 | 往 json 文件里添加不存在的事件/快捷键 | 往 json 文件里添加不存在的事件/快捷键，并打印输出是否注册成功 | 显示不存在，未注册 | 通过 |
| 4 | 一个功能函数 | 把 json 文件改 | 都可以触发，没 | 通过 |

| | | | | |
|---|---|---|---|---|
| | 对应两个快捷键 | 一改，一个功能函数对应两个快捷键，并输出所有键值对 | 有对此进行限制 | |
| 5 | 一个快捷键绑定两个功能 | 把 json 文件改一改，一个功能函数对应两个快捷键，并输出所有键值对 | 按 json 顺序，后一个会覆盖前一个 | 通过 |
| 6 | 测试并发同时修改文件与读取文件进行调用 | 在修改文件时，进行快捷键的加载 | 无响应，线程安全 | 通过 |

**功能点：工作区，目前功能不完善（手动测试）**

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果 |
|---|---|---|---|---|
| 1 | 正常切换到一个工作区 | 按 crtl+数字键，并打印输出该 vector 下存储的二叉树根节点是否为新 | 切换到对应工作区，一个崭新的工作区 | 通过 |
| 2 | 切换到一个新的工作区后，返回原本的工作区 | 按 crtl+原本的工作区数字，并打印输出原本的二叉树的各个节点查看是否一致 | 返回到原本工作区且上下文保存 | 通过 |
| 3 | 连续切换工作区 | 连续按 crtl+数字键，并不打断打印输出 | 正常切换到对应工作区 | 通过 |
| 4 | | | | |

TODO:待实现功能点，动态扩展工作区数量，把工作区直接映射到物理屏幕 output 上

**功能点：删除窗口（手动测试）**

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果） |
|---|---|---|---|---|
| 1 | 正常关闭平铺窗口 | 点击“x“这个按钮，并打印容器树查看是否删除对应节点，以及是否把另一个节点升级 | 该窗口关闭，把该窗口的第一个父容器下的另一个窗口升级 | 通过 |
| 2 | 连续关闭多个平铺窗口 | 连续点击”X” | 连续关闭窗口，正常升级和关闭 | 通过 |
| 3 | 关闭浮动窗口 | 关闭浮动窗口时尝试关闭 | 关闭时禁用了浮动窗口 | 通过 |

**功能点：浮动机制(手动测试)**

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果 |
|---|---|---|---|---|
| 1 | 把窗口切换为浮动 | 按住alt选中一个窗口，输出它所在的层，查看是否与应用层不一致 | 该窗口切换为浮动状态 | 通过 |
| 2 | 浮动窗口的大幅度拖拽 | 按住alt选择一个窗口自由拖拽 | 该窗口跟随鼠标拖拽 | 通过 |
| 3 | 浮动窗口拖拽到总窗口外围 | 按住alt选择一个窗口往窗口外拖 | 当超过一定范围，报错超出屏幕范围 | 通过 |
| 4 | 按住alt后，选中“X”按钮，查看是否具有冲突 | 按住alt点击窗口的‘X”按钮进行测试 | 按住alt再点击x按钮应该依然是切换到浮动状态，不会关闭窗口 | 通过 |

**功能点：动画切换机制**

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果） |
|---|---|---|---|---|
| 1 | 切换工作区时动画是否正常运行 | 按 crtl+任意数字键 | 切换到对应工作区且动画流程 | 通过 |
| 2 | 连续切换工作区查看动画切换是否正常 | 连续按 crtl+任意数字键 | 每次切换的时候都很流畅 | 通过 |
| 3 | 切换动画时，有无透明特效 | 切换工作区时，输出每一帧渲染时的参数，进行查看 | 透明度按正常逻辑进行变换 | 通过 |

**功能点：键盘输入**

| 编号 | 测试点 | 操作步骤 | 预期结果 | 结果） |
|---|---|---|---|---|
| 1 | 正常输入字符 | 输入字符，脚本化输入随机字母 | 正常输出 | 通过 |
| 2 | 输入大写字符 | 按住 shift 键，再输入字母 | 可正常输出大写字符 | 通过 |
| 3 | 输入多个字符 | 输入多个字符 | 可正常流畅的输出 | 通过 |

# 五．内存安全测试：

我们使用 valgrind 框架去测试内存泄漏：

**测试参数**

```
valgrind --tool=memcheck \
  --leak-check=full \
  --show-leak-kinds=all \
  --track-origins=yes \
  --errors-for-leak-kinds=definite \
  --error-exitcode=42 \
  --log-file=/home/zero/tiley/tmp/valgrind.%p.log \
  timeout -s INT -k 5s 60s \
  ./build/tiley --debug
```

**测试期间执行了以下命令：新建窗口 5 次，把一个窗口切换为浮动状态并来回拖拽，切换工作区**

**测试结果：**

```
==6832== Memcheck, a memory error detector
==6832== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==6832== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==6832== Command: timeout -s INT -k 5s 60s ./build/tiley --debug
==6832== Parent PID: 6571
==6832==
==6832==
==6832== HEAP SUMMARY:
==6832==       in use at exit: 0 bytes in 0 blocks
==6832==    total heap usage: 31 allocs, 31 frees, 3,997 bytes allocated
==6832==
==6832== All heap blocks were freed -- no leaks are possible
==6832==
==6832== For lists of detected and suppressed errors, rerun with: -s
==6832== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

**内存测试结果分析：**

1. in use at exit: 0 bytes in 0 blocks 由该结果可得程序在结束时成功释放了所有分配的内存，没有内存泄漏。
2. total heap usage: 31 allocs, 31 frees, 3,997 bytes allocate 由该结果可得，·

  31allocs：程序总共进行了 31 次内存分配。
·31frees：程序进行了 31 次内存释放。
·3,997 bytes allocated：程序总共分配了 3997 字节的内存

3All heap blocks were freed -- no leaks are possible，由该结果可得所有

分配的堆内存都已经被释放了，因此没有内存泄漏。也就是在程序退出时，所有分配的内存块都已经回收，可见完全遵守 rall 机制。
4. ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)由该结果可得程序在执行的过程中，valgrind 没有发现内存有错误（未初始化的内存访问，越界访问，重复释放内存）。

# 六．性能测试:

在程序运行后，监控于 output 里绘制函数前后统计帧数 cpu 使用率等，结果如下：

**性能输出结果：**

FPS: 63.829, CPU(s): 0.073472, CPU/FPS Ratio: 0.00115107, Memory(MB): 82.3711, Avg Render Time (ms): 0.301441
FPS: 48.6637, CPU(s): 0.116523, CPU/FPS Ratio: 0.00239445, Memory(MB): 86.5078, Avg Render Time (ms): 0.238361
FPS: 54.4138, CPU(s): 0.147355, CPU/FPS Ratio: 0.00270805, Memory(MB): 86.9023, Avg Render Time (ms): 0.232899
FPS: 54.056, CPU(s): 0.181795, CPU/FPS Ratio: 0.00336309, Memory(MB): 86.9023, Avg Render Time (ms): 0.219961
FPS: 57.3488, CPU(s): 0.223914, CPU/FPS Ratio: 0.00390442, Memory(MB): 86.9023, Avg Render Time (ms): 0.215918
FPS: 52.9785, CPU(s): 0.268827, CPU/FPS Ratio: 0.00507427, Memory(MB): 86.9023, Avg Render Time (ms): 0.222304
FPS: 51.5268, CPU(s): 0.313388, CPU/FPS Ratio: 0.00608204, Memory(MB): 86.9023, Avg Render Time (ms): 0.230559
FPS: 51.7445, CPU(s): 0.353033, CPU/FPS Ratio: 0.00682262, Memory(MB): 86.9023, Avg Render Time (ms): 0.231117
FPS: 54.5303, CPU(s): 0.384111, CPU/FPS Ratio: 0.007044, Memory(MB): 86.9023, Avg Render Time (ms): 0.234071
FPS: 55.5936, CPU(s): 0.421318, CPU/FPS Ratio: 0.00757853, Memory(MB): 86.9023, Avg Render Time (ms): 0.234814
FPS: 55.6894, CPU(s): 0.473264, CPU/FPS Ratio: 0.00849828, Memory(MB): 86.9023, Avg Render Time (ms): 0.231909
FPS: 56.7207, CPU(s): 0.519639, CPU/FPS Ratio: 0.00916136, Memory(MB): 86.9023, Avg Render Time (ms): 0.22876
FPS: 57.0355, CPU(s): 0.565628, CPU/FPS Ratio: 0.00991713, Memory(MB): 86.9023, Avg Render Time (ms): 0.228201
FPS: 57.994, CPU(s): 0.617992, CPU/FPS Ratio: 0.0106561, Memory(MB): 86.9023, Avg Render Time (ms): 0.227523
FPS: 58.4275, CPU(s): 0.712113, CPU/FPS Ratio: 0.012188, Memory(MB): 86.9023, Avg Render Time (ms): 0.233687

FPS: 59.4478, CPU(s): 0.755387, CPU/FPS Ratio: 0.0127067, Memory(MB): 86.9023, Avg Render Time (ms): 0.23239
FPS: 60.0757, CPU(s): 0.802151, CPU/FPS Ratio: 0.0133523, Memory(MB): 86.9023, Avg Render Time (ms): 0.230525
FPS: 60.0732, CPU(s): 0.860792, CPU/FPS Ratio: 0.014329, Memory(MB): 86.9023, Avg Render Time (ms): 0.226903
FPS: 60.4611, CPU(s): 0.929493, CPU/FPS Ratio: 0.0153734, Memory(MB): 86.9023, Avg Render Time (ms): 0.228768
FPS: 59.312, CPU(s): 0.965708, CPU/FPS Ratio: 0.0162818, Memory(MB): 86.9023, Avg Render Time (ms): 0.227967
FPS: 60.6607, CPU(s): 1.00679, CPU/FPS Ratio: 0.0165971, Memory(MB): 86.9023, Avg Render Time (ms): 0.227587
FPS: 59.7579, CPU(s): 1.02954, CPU/FPS Ratio: 0.0172285, Memory(MB): 86.9023, Avg Render Time (ms): 0.224702
FPS: 60.3235, CPU(s): 1.05697, CPU/FPS Ratio: 0.0175217, Memory(MB): 86.9023, Avg Render Time (ms): 0.223682
FPS: 60.5631, CPU(s): 1.0879, CPU/FPS Ratio: 0.0179632, Memory(MB): 87.2617, Avg Render Time (ms): 0.222102

**性能分析如下：**

1. FPS（帧率）
范围：最低 48.7 FPS，最高 63.8 FPS，大部分集中在 55–61 FPS。
趋势：初始略高（~64 FPS），很快稳定在 ~59–60 FPS，说明系统进入了接近刷新率锁定（60Hz）的状态。
（在 VirtualBox 里，渲染通常受我们的虚拟机的显示刷新/虚拟显卡驱动限制，所以帧率波动是同步机制的结果。）

2. 渲染耗时（Avg Render Time）
均值：约 0.22–0.30 ms/帧。
特点：极短（远小于理论帧预算 16.7ms/60Hz），说明：
CPU 渲染逻辑很轻。
大部分时间其实是等待 vsync 或虚拟显示刷新，不是耗在渲染本身。
趋势：随着运行，渲染平均耗时略下降并趋稳（从 0.30ms → ~0.22ms），表现稳定。

3. CPU 使用
累计 CPU 时间从 0.07s 增加到 1.08s，对应测试过程逐步增长。
CPU/FPS Ratio 从 ~0.0011 增加到 ~0.018，说明“每帧的累计 CPU 成本”在慢慢拉高。
换算：大约 0.7–0.8 ms CPU 时间/帧，相当于单核利用率 ~4–5%，非常轻量。
解释：CPU 占用非常低，说明我们的 tiely 平铺管理器框架在 CPU 层面几乎没压力。

**4 内存占用**
初始 82 MB，稳定在 86–87 MB。
趋势：没有持续上涨，说明没有明显内存泄漏。
内存波动小，常驻集合理。

**5. 总体表现**
帧率稳定：接近 60Hz，表明虚拟环境下渲染循环可靠，没有明显掉帧。
CPU 占用低：单核几乎无压力（<10%），可以预期在真机上表现更好。
渲染耗时极短：表明渲染逻辑轻量，主要瓶颈是 vsync/显示刷新，而非代码本身。
内存稳定：运行过程中基本保持 ~87MB，无泄漏迹象。

**性能总结：**

测试结果说明 Tiley 在 VirtualBox + Ubuntu 环境下运行性能非常轻量，CPU 与内存开销都小，帧率稳定在接近显示器刷新率水平。

# 七．稳定性测试：

**目标：**
连续运行 8 个小时，验证是否会出现崩溃，死锁，资源泄漏。且关键性能比如 FPS 帧率，内存在合理范围之内。
**测试方法：**
以每 10 秒为一个小周期，循环执行如下操作：
1.新建/关闭窗口：窗口 5 个，关闭 3 个。
2.布局操作：平铺状态与浮动状态来回切换（把窗口在容器树来回取出放入）
3.焦点输入：焦点切换。输出鼠标坐标
4.工作区来回切换，在工作区 1，2，3 来回切换。
5.在 3 张壁纸内循环切换
6.热重载，随机切换两个函数与其对应的快捷键。

**结果：**
挂 8 个小时运行没问题，并且途中测试一轮也没问题，整体非常稳定。