《Rust程序设计》课程实践报告

小组成员：安锐博，斯文

项目： Partrick's Parabox 简易版

GitHub仓库地址：Patrick-s-Parabox-Rust

概览

我们用 Rust 语言重制了推箱子解谜游戏 Partrick's Parabox 的简易版，共1000+行 Rust 代码。它基于 Bevy 游戏引擎的 0.16.1 版本开发。
我们为这个项目做了以下工作：

- 阅读 Bevy 游戏引擎的相关文档，学习了其中基本的接口使用方法和 ECS (Entity, Compenent, System)游戏开发逻辑。
- 完成游戏的主体设计和开发，包括游戏的基本逻辑、图形界面、音效等。

背景： Bevy 游戏引擎

Bevy 是一个开源的 Rust 游戏引擎，它为开发者提供了一个跨平台的框架，用于开发游戏。 Bevy 引擎的一大特色是 ECS 机制； ECS 是指 Entity-Component-System （实体-组件-系统）。 Entity 是实体或对象，由 Component 构成；每一个 Component 都是一个 struct （仅含一个整型参数）； System 则是指程序的执行机制或函数， System 和 Component 的关系是通过 World 和 Entity 索引建立的。 Bevy 引擎内置了很多 system 的 plugin ，开发者可以通过 plugin 自由组合并从外部模块接入所需要的功能，来设计玩法、界面并与计算机的键盘、鼠标等设备交互。

代码结构概览

我们的游戏有以下几个界面：

- 主界面
- 关卡选择界面
- 游戏界面
- 胜利结算界面

接下来的报告将围绕这些界面中的关键代码展开。

主界面

主菜单和关卡选择界面在 `menu_plugin` 中实现，这个 `plugin` 向 `App` 中添加了若干个 `system` ，用来控制进入特定状态时各菜单的渲染、点击按钮时的交互以及状态转移前屏幕元素的清除，主要代码如下：

```
pub fn menu_plugin(app: &mut App) {
    app
        .init_state::<MenuState>()
        .add_systems(OnEnter(GameState::Menu), menu_setup)
        .add_systems(OnEnter(MenuState::Main), main_menu_setup)
        .add_systems(OnExit(MenuState::Main), despawn_screen::<OnMainMenuScreen>)
        .add_systems(Update, (start_button.run_if(in_state(MenuState::Main)),))
        .add_systems(OnEnter(MenuState::Levels), level_select_menu_setup)
        .add_systems(OnExit(MenuState::Levels), despawn_screen::<OnLevelSelectScreen>)
        .add_systems(Update, level_button.run_if(in_state(MenuState::Levels)))
        .add_systems(Update, (menu_action, button_system).run_if(in_state(GameState::Menu)))
        .add_systems(Update, (menu_action, button_system).run_if(in_state(GameState::LevelSelect
}
```

具体来说，菜单是一个 `State Machine` ，我们在 `menu.rs` 中定义了一个 `MenuState` 的 `enum` ：

```
#[derive(Clone, Copy, Default, Eq, PartialEq, Debug, Hash, States)]
pub enum MenuState {
    Main,
    Levels,
    #[default]
    Disabled,
}
```

其中包括了主菜单和关卡选择菜单，默认状态是禁用；主菜单样式如下：

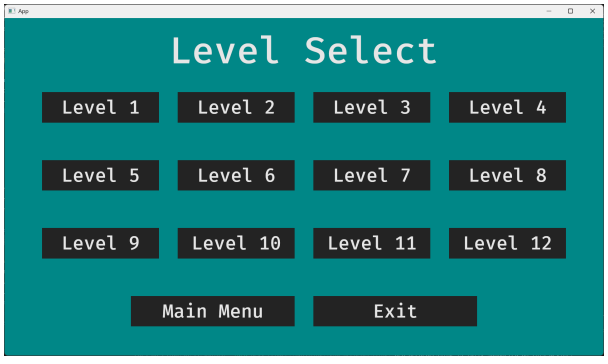
System `main_menu_setup` 会在 `Menustate` 变为 `Main` 状态时运行，利用 `commands.spawn` 和一个 `asset_server` (用来加载音频等多媒体素材)，在屏幕中央渲染出菜单界面以及其中的各个按钮。菜单界面的层次结构是利用 `spawn bundle` 时的 parent-children 特性实现，在 `spawn` 一个 `Bundle` 后，可以用 `.with_children(|parent| {...})` 传入一个 `closure` ， `closure` 中可以继续用 `parent.spawn` 生成子元素。例如上图中，最底层的背景板是一个 `Node` 用于设置背景颜色和BGM;它的儿子是另一个 `Node` 用于居中显示，该 `Node` 有五个儿子，分别是一个 `Text` "Patrick's Parabox" 和两个 `Button` , 而两个 `Button` 各有一个儿子 `Text` 用来写对应的文字。这种树形结构使得菜单的组织非常清晰。

`Button` 被点击之后的跳转也是通过若干个 `system` 实现，他们监听按钮交互事件并对 `MenuState` 和 `GameState` 作出相应的修改，从而使游戏在 `State Machine` 上移动。例如 `menu_action` 是对主菜单各按钮点击交互的处理：

```
pub fn menu_action(...) {
    ...
    match menu_button_action {
        MenuButtonAction::Quit => {
            app_exit_events.write(AppExit::Success);
        }
        MenuButtonAction::SelectLevel => {
            game_state.set(GameState::LevelSelect);
            menu_state.set(MenuState::Levels);
        }
        MenuButtonAction::BackToMainMenu => {
            game_state.set(GameState::Menu);
            menu_state.set(MenuState::Main);
        }
    }...
}
```

关卡选择界面

在主菜单中点击 `Start` 会进入关卡选择界面。



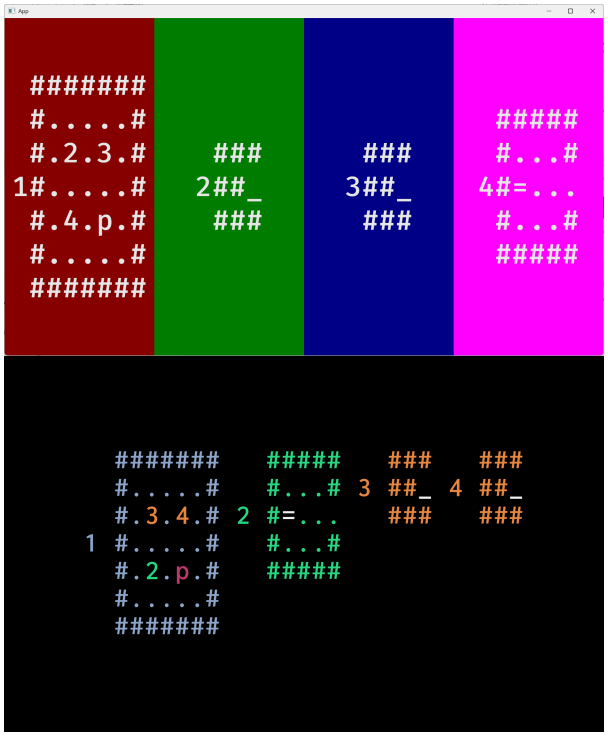
关卡的实现机制是定义了一个全局的 resource ：

```
#[derive(Resource, Debug, Component, PartialEq, Eq, Clone, Copy)]
struct Level(i32);
```

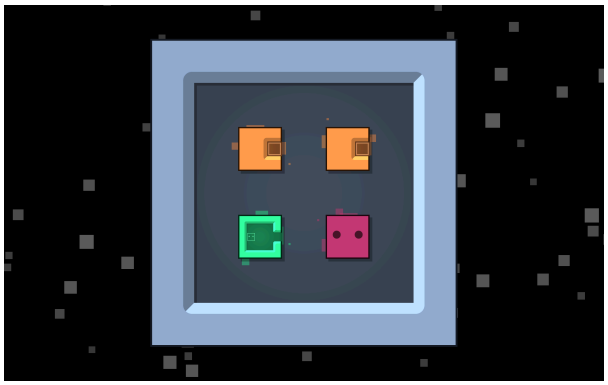
在菜单中用上文介绍的方法和思路渲染出关卡选择的菜单，并设计一个 level_button 的 system ，用于进入不同的关卡；在点击选关按钮时我们会修改 Level 的参数，来选择要进入的关卡。

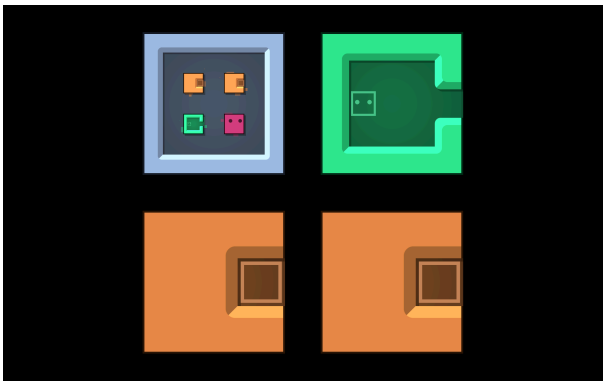
游戏界面

游戏界面中，为了方便画面渲染，我们参考了原游戏中的字符风格界面，用 p 表示玩家， b 表示箱子， # 表示墙， . 表示空格，数字表示 Parabox ，也就是游戏中的容器； _ 表示箱子或 Parabox 的目标格， = 表示玩家的目标格；游戏胜利的判定即是所有目标格都被相应填满。。我们的游戏界面如下（右侧为原游戏的对比图）：



为更清晰的展示游戏机制，我们一并展示原游戏的另外两种界面风格：





关卡的展示通过传入实体类的 `resource` 即关卡的初始化 `LevelConfig` ， 我们在其中附着了其他方法...