Figure 1: https://bevyengine.org

一个简单的、令人耳目一新的数据驱动游戏引擎，基于 Rust 构建，永远免费和开源。

1. 概述

数据驱动
所有引擎和游戏逻辑均采用 Bevy ECS（一种自定义实体组件系统）
- **快速：** 大规模并行和缓存友好。根据一些基准测试，这是最快的 ECS
- **简单：** 组件是 Rust 结构，系统是 Rust 函数
- **功能：** 查询、全局资源、本地资源、变化检测、无锁并行调度程序

```rust
#[derive(Component)]
struct Player;

fn system(
  q: Query<(Entity, &Player)>
) {
}
```

2D 渲染器
为游戏和应用程序渲染实时 2D 图形
- **功能：** 精灵表、动态纹理图集、相机、纹理和材质
- **可扩展：** 自定义着色器、材质和渲染管线
- **核心：** 建立在 Bevy 的渲染图上

3D 渲染器
现代完美适的 3D 渲染器
- **特点：** 灯光、阴影、相机、网格、纹理、材质、gltf 加载
- **可扩展：** 自定义着色器、材质和渲染管线
- **核心：** 建立在 Bevy 的渲染图上

动画
强大的动画系统
- 由基于 ECS 的关节 API 驱动的骨骼绑定动画
- 通过平滑混合来同时播放多个动画
- 使用混合形状/变形目标直接为顶点设置动画
- 从 GLTF 文件导入动画

跨平台
支持所有主流平台
- Windows、MacOS、Linux、Web、IOS、Android

免费且开源
由开发者社区打造并服务于开发者社区的引擎
- 100% 免费，永远免费
- 根据宽松的 MIT 或 Apache 2.0 许可证开放源代码
- 无合同
- 无许可费用
- 无销售分成

2. Bevy ECS

2.1 ECS 的概念与组成

- Entity（实体）：实体是一个唯一标识符，用于关联一组组件，实体本身不包含任何数据和行为。

```rust
struct Entity {
  index: u32,
  generation: NonZeroU32,
}
```

- Component（组件）：组件是数据的容器，不包含行为逻辑，每个组件只负责存储一类数据。

```rust
#[derive(Component)]
struct Player {
  health: u32, // 生命值
  magic: u32, // 魔法值
}
```

- System（系统）：系统是行为逻辑，负责处理特定类型的组件，系统会遍历所有包含相关组件的实体。

```rust
fn player_info(players: Query<&Player>) {
  for &Player { health, magic} in &players {
    info!("health: {health}, magic: {magic}");
  }
}
```

2.2 ECS 的优势

- 解耦：通过分离数据（组件）和行为（系统），ECS 降低了对象之间的耦合度，使得代码更易于维护和拓展。
- 性能优化：ECS 可以更高效的利用缓存和并行处理，特别适合处理有大量对象的场景。
- 灵活性：添加新功能只需要增加新的组件和系统，不需要修改现有的组件和系统。

3. 安装 LLD 链接器

3.1 在 Windows 操作系统上安装 LLD 链接器

- 执行：`cargo install -f cargo-binutils`

```
  Installing C:\Users\desig\.cargo\bin\rust-cov.exe
  Installing C:\Users\desig\.cargo\bin\rust-ld.exe
  Installing C:\Users\desig\.cargo\bin\rust-lld.exe
  Installing C:\Users\desig\.cargo\bin\rust-nm.exe
  Installing C:\Users\desig\.cargo\bin\rust-objcopy.exe
  Installing C:\Users\desig\.cargo\bin\rust-objdump.exe
  Installing C:\Users\desig\.cargo\bin\rust-profdata.exe
  Installing C:\Users\desig\.cargo\bin\rust-readobj.exe
  Installing C:\Users\desig\.cargo\bin\rust-size.exe
  Installing C:\Users\desig\.cargo\bin\rust-strip.exe
   Installed package `cargo-binutils v0.3.6` (executables `cargo-cov.exe`, `cargo-nm.exe`, `cargo-objcopy.exe`, `cargo-objdump.exe`, `cargo-profdata.exe`, `cargo-readobj.exe`, `cargo-size.exe`, `cargo-strip.exe`, `rust-ar.exe`, `rust-cov.exe`, `rust-ld.exe`, `rust-lld.exe`, `rust-nm.exe`, `rust-objcopy.exe`, `rust-objdump.exe`, `rust-profdata.exe`, `rust-readobj.exe`, `rust-size.exe`, `rust-strip.exe`)
```

- 执行：`rustup component add llvm-tools-preview`

```
info: downloading component 'llvm-tools'
info: installing component 'llvm-tools'
 30.5 MiB /  30.5 MiB (100 %)  15.9 MiB/s in  1s ETA:  0s
```

3.2 在其它操作系统上安装 LLD 链接器

- Ubuntu: `sudo apt-get install lld clang`
- Fedora: `sudo dnf install lld clang`
- arch: `sudo pacman -S lld clang`
- MacOS: 在 MacOS 上，默认系统链接器 ld-prime 比 LLD 更快

4. 创建 Bevy 项目

- 执行：`cargo new bevy_games`
- 在 bevy_games 中新建 .cargo 文件夹，在 .cargo 文件夹中新建 config.toml 文件
- 在 config.toml 中添加

```toml
# 使用 LLD 链接器，如果不使用 LLD 可以跳过
[target.x86_64-pc-windows-msvc] # Windows 平台
linker = "rust-lld.exe"

[target.x86_64-unknown-linux-gnu] # Linux  平台
linker = "clang"
rustflags = ["-C", "link-arg=-fuse-ld=lld"]
```

- 在 Cargo.toml 中添加

```toml
# 添加 Bevy 依赖
[dependencies]
bevy = { version = "0.14", features = [
  "dynamic_linking",  # 加快链接速度，在 Windows 上如果没有使用 LLD 链接器，可能出错
  "bevy_dev_tools", # 内置的开发工具
  "shader_format_spirv", # 接受 spirv 作为 shader
]}

# Rust 优化有 0-3 四个级别，开发模式下默认使用 0 级优化
[profile.dev] # 项目在开发模式下使用 1 级优化
opt-level = 1

[profile.dev.package."*"] # 项目依赖在开发模式下使用 3 级优化
opt-level = 3
```