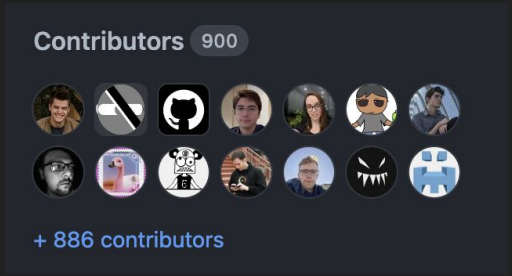
Entity **C**omponent **S**ystem

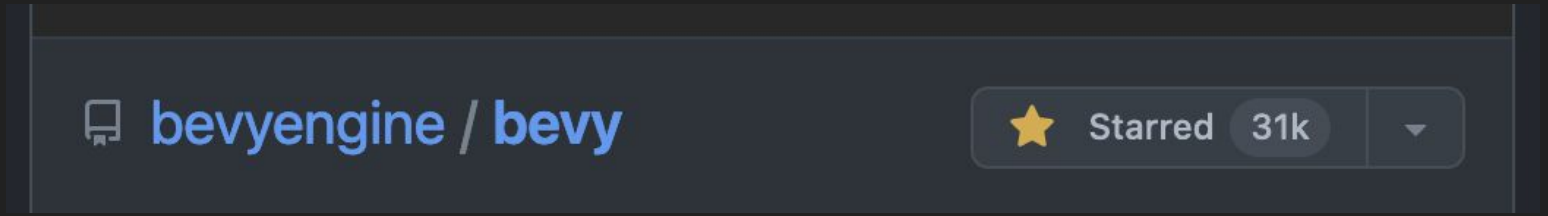
Power tool for data oriented applications

Rust Syd 2024-02-27

bevy_ecs

- Purpose build ECS for the Bevy game engine
- Ergonomic
- Fast
- Parallel
- Active Development

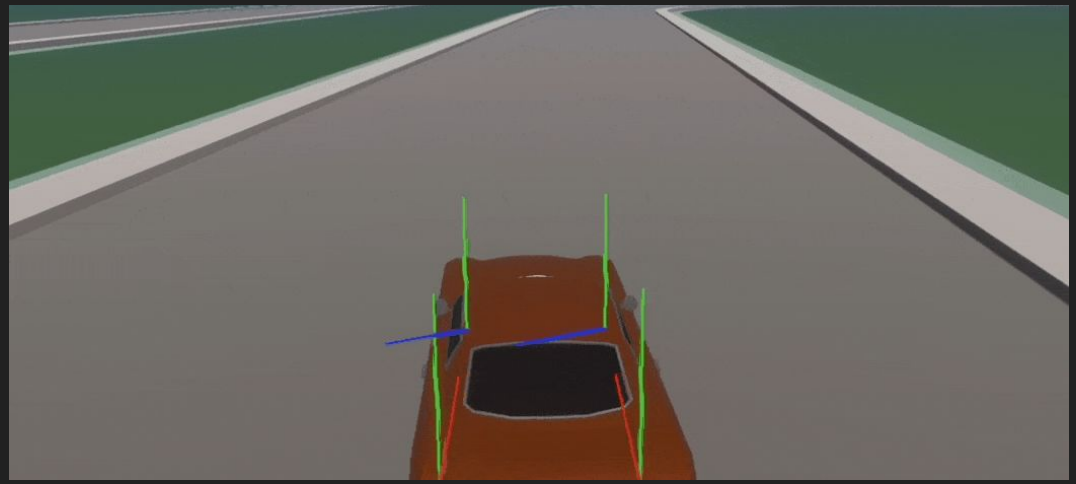





Level 3
[H] for help
AP 0
HP 000
[1]
[2]


Left click to dig ($64)


STONE: 0
LEATHER: 0
COPPER: 0
IRON: 0
STEEL: 0
FOOD: 0
MEAD: 0
HERBS: 0
GEMS: 0
RUNES: 0
PLAYER
Ashwood Bow
Iron Sword




Ground Plane
Money: 50
Food: 0
Wood: 0
House
Barracks
Guard Tower

Entities

```
struct Entity(u64);
```

Components

```
struct Name(String);

struct CurrentScore(i32);

struct HighScore(i32);
```

Systems

```
fn print_names(query: Query<&Name>) {
    for name in query.iter() {
        println!("{name:?}");
    }
}
```

	A	B	C	D
1	**Entity**	**Name**	**CurrentScore**	**HighScore**
2	1	Bob	0	13
3	2	Alice		
4	3	Ferris	0	6
5	4	Roger		

	A	B	C	D
1	**Entity**	**Name**	**CurrentScore**	**HighScore**
2	1	Bob	0	13
3	3	Ferris	0	6

	A	B
1	**Entity**	**Name**
2	2	Alice
3	4	Roger

```rust
fn give_score(mut query: Query<&mut CurrentScore>) {
    for mut score in query.iter_mut() {
        match random_f32() {
            x if x > 0.7 => { // 30%
                score.0 += 1;
            }
            x if x < 0.1 => { // 10%
                score.0 = 0;
            }
            _ => {}           // 60%
        }
    }
}
```

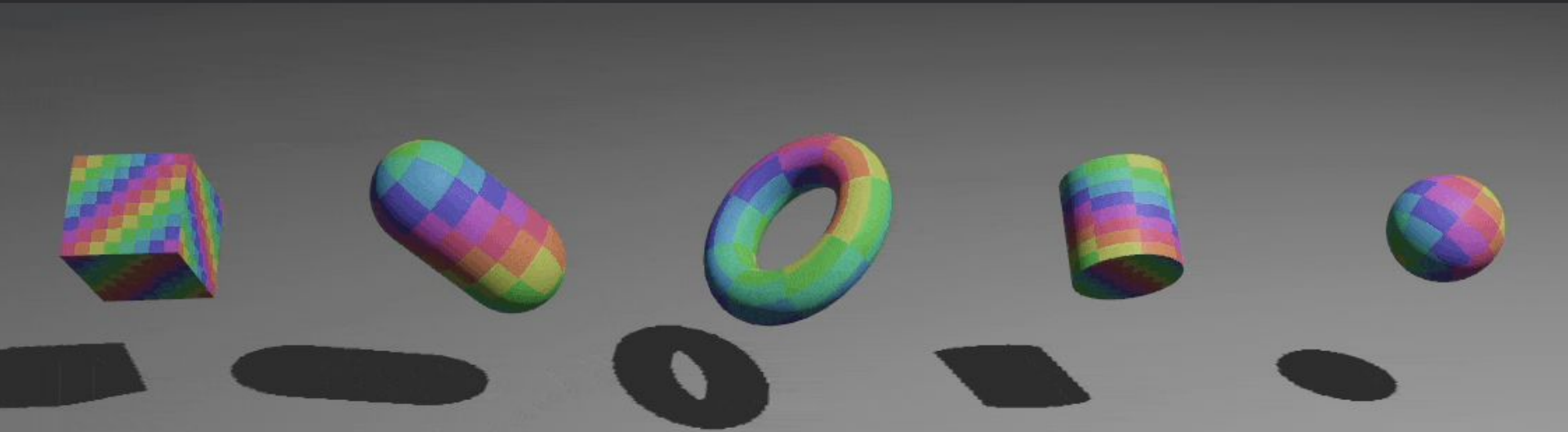
```rust
fn update_highscores(mut query: Query<
    (&CurrentScore, &mut HighScore),
    Changed<CurrentScore>
>) {
    for (current, mut high) in query.iter_mut() {
        if current.0 > high.0 {
            high.0 = current.0;
        }
    }
}
```

```rust
fn print_highest_score(query: Query<(&HighScore, &Name)>) {
    let highest_score = query
        .iter()
        .reduce(|(score, name), (other_score, other_name)| {
            if other_score.0 > score.0 {
                (other_score, other_name)
            } else {
                (score, name)
            }
        });
    if let Some((score, name)) = highest_score {
        println!("Current High Score {}: {}", name, score.0);
    }
}
```

```rust
let mut world = World::new();
world.spawn((Name::new("Bob"), CurrentScore(0), HighScore(0)));
world.spawn((Name::new("Alice"), CurrentScore(0), HighScore(0)));

let mut update = Schedule::default();
update.add_systems((
    give_score,
    update_highscores,
    print_highest_score,
).chain());

loop {
    update.run(&mut world);
}
```

No lifetimes!

```rust
fn main() {
    App::new().add_plugins(MySimulation).run();
}
struct MySimulation;
impl Plugin for MySimulation {
    fn build(&self, app: &mut App) {
        app.add_systems(Startup, setup);
        app.add_systems(
            Update,
            (give_score, update_highscores, print_highest_score).chain(),
        );
    }
}
fn setup(world: &mut World) {
    world.spawn((Name::new("Bob"), CurrentScore(0), HighScore(0)));
    world.spawn((Name::new("Alice"), CurrentScore(0), HighScore(0)));
}
```



```rust
fn main() {
    App::new().add_plugins((
        MySimulation,
        bevy_render::RenderPlugin,
    )).run();
}
```

```
for shape in shapes.into_iter() {
    world.spawn((
        Name::new("Alice"), CurrentScore(0), HighScore(0),
        PbrBundle {
            mesh: shape,
            material: debug_material.clone(),
            transform: Transform::from_xyz(x, y, z),
            ..default()
        },
    ));
}
```

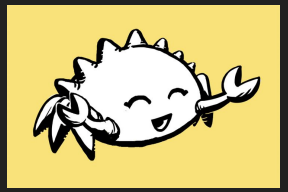
```rust
fn rotate(
    mut query: Query<(&mut Transform, &CurrentScore)>,
    time: Res<Time>
) {
    for (mut transform, score) in &mut query {
        transform.rotate_y(time.delta_seconds() * score.0 as f32);
    }
}
```

```
App::new()
    .add_plugins((DefaultPlugins, bevy_xpbd_3d::PhysicsPlugins));

world.spawn((
    RigidBody::Dynamic,
    Collider::cuboid(1.0, 1.0, 1.0),
    PbrBundle {
        mesh: meshes.add(Cuboid::default()),
        material: materials.add(Color::rgb(0.8, 0.7, 0.6)),
        transform: Transform::from_xyz(0.0, 4.0, 0.0),
        ..default()
    },
));
```

```rust
let mut group = PluginGroupBuilder::start::<Self>();
group = group
    .add(bevy_log::LogPlugin::default())
    .add(bevy_core::TaskPoolPlugin::default())
    .add(bevy_core::TypeRegistrationPlugin)
    .add(bevy_core::FrameCountPlugin)
    .add(bevy_time::TimePlugin)
    .add(bevy_transform::TransformPlugin)
    .add(bevy_hierarchy::HierarchyPlugin)
    .add(bevy_diagnostic::DiagnosticsPlugin)
    .add(bevy_input::InputPlugin)
    .add(bevy_window::WindowPlugin::default())
    .add(bevy_a11y::AccessibilityPlugin);

#[cfg(feature = "bevy_asset")]
{
    group = group.add(bevy_asset::AssetPlugin::default());
}

#[cfg(feature = "bevy_scene")]
{
    group = group.add(bevy_scene::ScenePlugin);
}

#[cfg(feature = "bevy_winit")]
{
    group = group.add(bevy_winit::WinitPlugin::default());
}
```

```rust
#[cfg(feature = "bevy_render")]
{
    group = group
        .add(bevy_render::RenderPlugin::default())
        // NOTE: Load this after renderer initialization so that it knows abou
        // compressed texture formats
        .add(bevy_render::texture::ImagePlugin::default());

    #[cfg(all(not(target_arch = "wasm32"), feature = "multi-threaded"))]
    {
        group = group.add(bevy_render::pipelined_rendering::PipelinedRendering
    }
}

#[cfg(feature = "bevy_core_pipeline")]
{
    group = group.add(bevy_core_pipeline::CorePipelinePlugin);
}

#[cfg(feature = "bevy_sprite")]
{
    group = group.add(bevy_sprite::SpritePlugin);
}

#[cfg(feature = "bevy_text")]
{
    group = group.add(bevy_text::TextPlugin);
}

#[cfg(feature = "bevy_ui")]
{
    group = group.add(bevy_ui::UiPlugin);
}
```

- ECS encourages decoupled design, separate data & logic
- ECS handles data organisation and lifetimes for perf & ergo
- Plugins give seamless composability without restructuring
- User code is engine code, vice versa
- Now you can write games in Rust!



Thank you!

mastodon.gamedev.place/@sleepytea
github.com/tbillington
@SleepyTeaGames

https://promethia-27.github.io/dependency_injection_like_bevy_from_scratch/introductions.html