BEVY UNIVERSITY

DEVELOPMENT ENVIRONMENT SETUP AND WELCOME

- Save time by doing the bevy setup if not already done.
- Then we will do the introduction.

INSTALLATION AND CONFIGURATION OF TOOLS

1. **Install Rust**: rustup, rustc, cargo, clippy, rustfmt.
2. **Configure Editor**: Set up your preferred editor with Rust Analyzer.
3. **Get bevy dependencies** according to your OS.
   https://bevyengine.org/learn/quick-start/getting-started/setup/#installing-os-dependencies
4. **Clone the workshop repository** from GitHub.
   https://github.com/uggla/bevy_university
5. **Compile** the project.

2 WORDS ABOUT US 1/2

- Stats
    - First name: Bastien
    - Last name: Sevajol
- Skills
    - Class: Software engineer
    - Latest Guilde: Biologic
    - Age of Experience: 16 years
    - Preferred weapon: Rust, Python
- Optional traits
    - Sleep in the forest
    - Play code and code games

2 WORDS ABOUT US 2/2

- Stats
  - First name: René (Uggla)
  - Last name: Ribaud
- Skills
  - Class: Software engineer
  - Previous Class: Solution architect (Cloud / Devops)
  - Latest Guilde: Red Hat
  - Game start: 1998
  - Preferred weapons: Rust / Python
  - Artefact: Openstack Nova
- Optional traits
  - Linux and FLOSS since 1995
  - Previously Ops, Dev today to produce my bugs

QUICK OVERVIEW OF RUST (OPTIONAL)

- Provide an alternative to C/C++ and also higher-level languages
- Multi paradigm language (imperative, functional, object oriented (not fully))
- Fast, safe, and efficient (modern)
- No garbage collector (good for games), ownership and borrow checker
- Dual license MIT and Apache v2.0
- First stable release May 15th, 2015

OVERVIEW OF BEVY

- Data-driven game engine built in Rust
- August 10, 2020 by Carter Anderson
- Dual license MIT and Apache v2.0
- Current version 0.15
- Not stable, breaking changes ~3 months and migration guides
- Fully based on the ECS pattern that encourages clean, decoupled designs
- Lots of features 2D, 3D, UI, Audio...
- Extensible with plugins
- Performant multi-threaded
- Multi platform (Windows, Linux, macOS, Web (Wasm), Android, iOS)

QUICK OVERVIEW OF THE PROJECT AND THE SESSION

- Goals of the Workshop:
  - First, we will follow the Bevy tutorial to understand the basics.
  - Next, we will explore additional concepts, equipping you with the tools to build your own game.
  - Finally we will produce a native and web (wasm) version
- What we will build together.
  - An mockup of Asteroids game.
- Session Overview
  - Provide an explanation of the tasks to be completed (via slides), showcasing the corresponding code. Engage actively by asking questions.
  - Work on the code yourself, and don't hesitate to ask for help if needed.
  - If you encounter any issues, no worries—you can always check out the branch for this specific section.

Paste screenshots of the game.

THE SHORTEST BEVY PROJECT

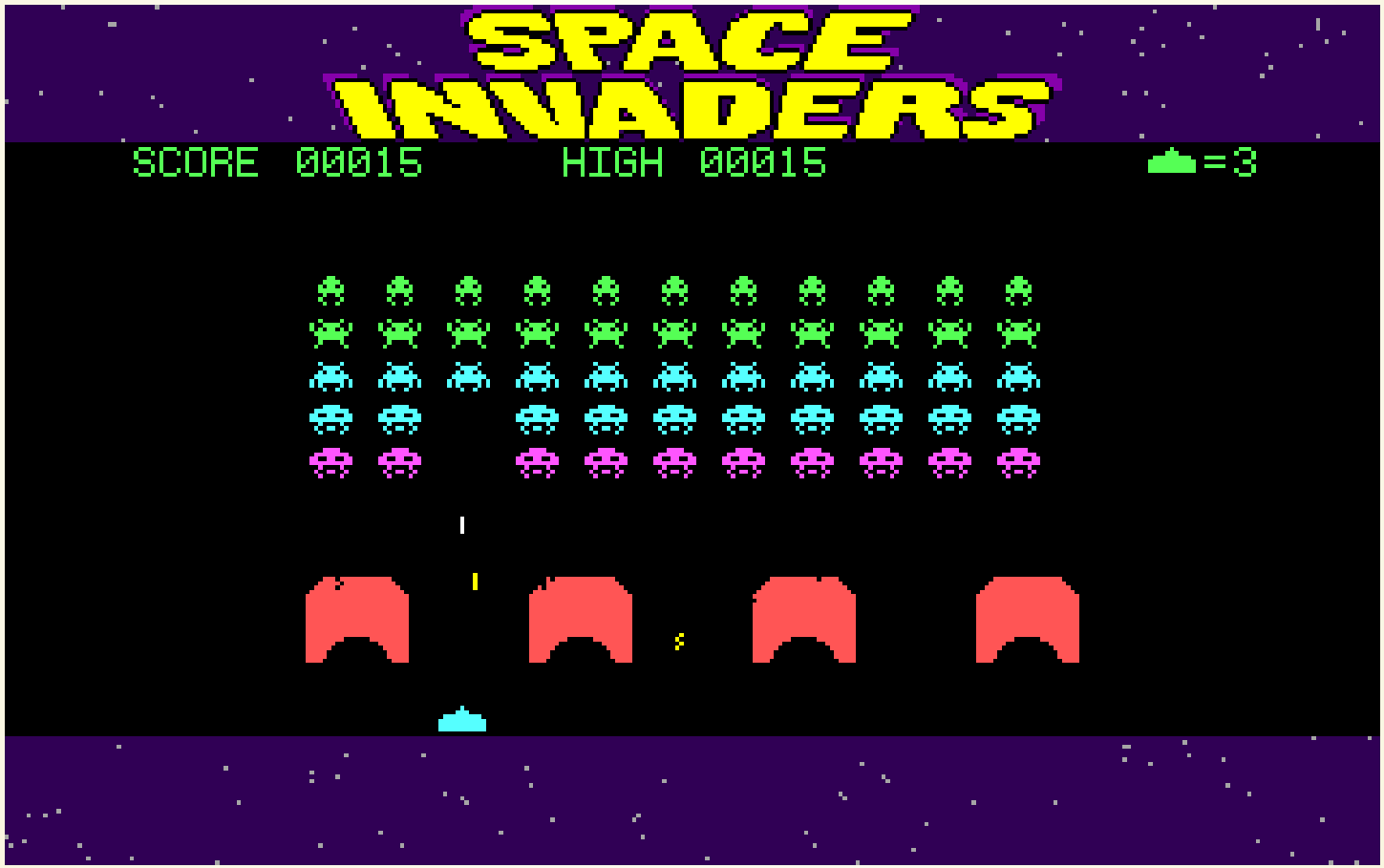
- Builder pattern

```rust
use bevy::prelude::*;

fn main() {
    App::new().run();
}
```

- Initializing resources in the World to store globally available data that we only need a single copy of.
- Adding systems to our Schedule, which can read and modify resources and our entities' components, according to our game logic.
- Importing other blocks of App-modifying code using Plugins.

INTRODUCTION TO THE ECS FRAMEWORK



ENTITY COMPONENT SYSTEM (ECS)

- **Entities**: Abstract game objects.

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

- **Components**: Data associated with entities.

```
#[derive(Component)]
struct Position {
    x: f32,
    y: f32,
}
```

- **Systems**: Logic that operates on entities and their components.

```
fn hello_world() {
    println!("hello world!");
}
```

ANALOGY TO SQL

ECS can be "compared" to an in memory SQL database. With only a big table

- **Entities**: Line in the table.
- **Components**: Columns in the table.
- **Systems**: Stored procedures in which we can query the table.

SYSTEMS AND SCHEDULING IN BEVY

INTRODUCTION TO BEVY SCHEDULER

- **Concept**: Coordinates system execution.
- **Practical Exercise**: Create and schedule systems in Bevy.

Branch	Files	Time
01-systems	src/main.rs	5 minutes

- The systems are run in parallel so they are not sequentially executed.
- But we can control the order of execution with methods `(after, before, chain)`.

MANIPULATING ENTITIES AND COMPONENTS IN BEVY

DECLARE ENTITIES AND ASSOCIATE COMPONENTS

- **Concept**: Create a Player component.
- Spawn an entity with the Player component.
- Various way of querying the component in a system.
- **Practical Exercise**:

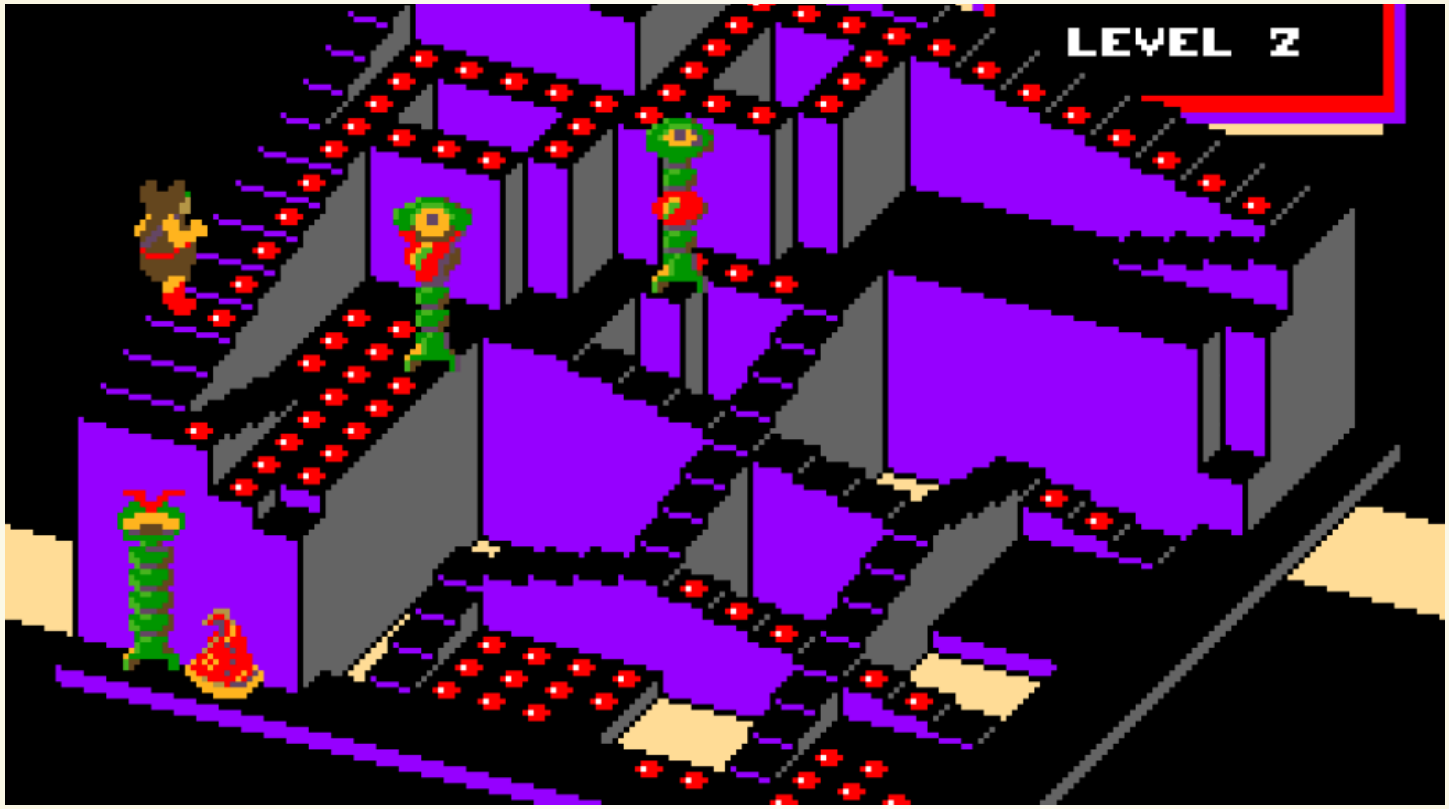
Branch	Files	Time
02-components_and_entity	src/main.rs	5 minutes

MUT COMPONENT(S) IN ENTITY(IES)

- **Concept**: Various way of querying the Player component in a system and change its name value.
- **Practical Exercise**:

Branch	Files	Time
03-mut_components_and_entities	src/main.rs	5 minutes

LOAD DEFAULT PLUGINS AND SET A CAMERA



LOAD DEFAULT PLUGINS

- **Concept**: Game loop and window.
- Load the default plugins.
- DefaultPlugins is a PluginGroup, a collection of plugins.
- Define windows size constants and set a window with appropriate size and title.
- my_second_system "update" is now in a 60fps game loop.
- **Practical Exercise**:

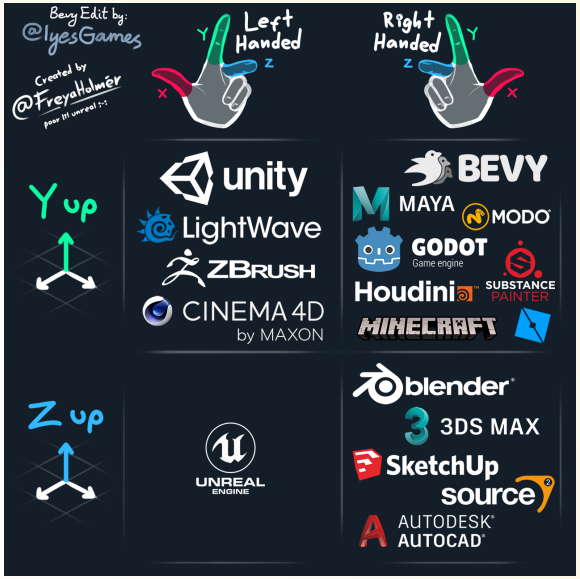
Branch	Files	Time
04-default_plugins	src/main.rs	5 minutes

SET A 2D CAMERA

- **Concept**: Camera and bundles.
- Modify first_system to spawn a Camera2dBundle or Camera2d (bevy 0.15).
- A bundle is a collection of components but is now not the preferred way to build components but still compatible.
- A required macro is now used to create dependencies between components.
- Coordinate (0,0) at the middle of the screen.
- Spawn a default sprite.
- **Practical Exercise**:

Branch	Files	Time
05-camera	src/main.rs	5 minutes

- Orientation



RESOURCES AND ASSET SERVER

DISPLAY OUR VESSEL

RESOURCES

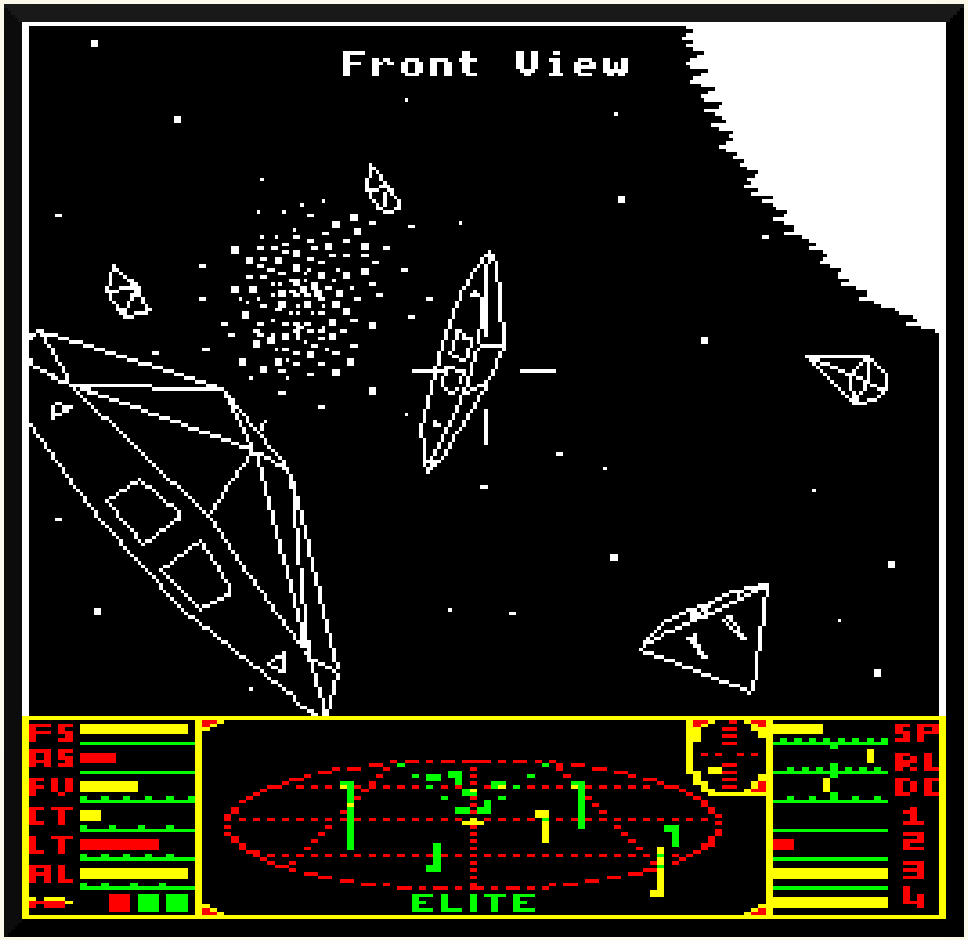
- **Definition**: Shared global data accessible by systems.
- Create a resource CurrentLevel and update it's value in the first_system.
- Resources need to be initialized in our application.
- Read the CurrentLevel resource in the second_system.

ASSET SERVER, IMAGE, TRANSFORM, DISPLAY OUR VESSEL

- Asset server is a builtin resource that read assets from the file system.
- Add an image to the sprite using the asset server.
- The image "sprites/player.png" can be loaded from the assets folder by default.
- Remove the sprite color red and add a transformation to reduce the size by 2.
- **Practical Exercise**:

Branch	Files	Time
06-resources	src/main.rs	5 minutes

MANAGE THE GAME STATES



STATES

- **Definition:** A state represents a specific mode or phase of your application, enabling systems to be selectively triggered based on the current state.
- While not essential for this mockup, states are highly beneficial. Implementing them from the start helps avoid a time-consuming rework later.
- States must derive the traits `Clone`, `Hash`, `Eq`, `PartialEq`, and `Default`.
- States need to be initialized within the application to be functional.
- Systems can be triggered or executed conditionally based on specific states.
- State information can be read using the `State` resource and updated with the `NextState` resource.
- **Practical Exercise:**

Branch	Files	Time
07-states	src/main.rs	5 minutes

CREATE YOUR OWN PLUGINS

PLUGINS

- **Definition:** A plugin is a modular unit of functionality that can be added to your application. Plugins help organize code, enable features, and set up the engine's systems, resources, and behaviors in a reusable and scalable way.
- Create a `states.rs` file, define a `StatePlugin`, and implement the `Plugin` trait.
- Refactor the code to move all state-related functionality to the `states.rs` file.
- **Caution:** Ensure the `GameState` struct is made public (`pub`).
- **Practical Exercise:**

Branch	Files	Time
08-plugins	src/main.rs src/states.rs	5 minutes

USER INPUTS

CREATE A SIMPLE MENU AND ENABLE VESSEL ROTATION

AT THIS POINT, WE HAVE COVERED ALMOST ALL THE FUNDAMENTALS, AND WE WILL INTRODUCE MORE CHANGES IN THE UPCOMING SECTIONS.

REFACTOR STATES PLUGIN TO DISPLAY A MENU

- Add a `Menu` component to tag menu-related entities.
- Add a system `display_menu` to create and display the menu when entering the `GameState::Menu` state.
- The menu consists of a parent entity with a `Node` component and a child entity with a `Text` component.
- Refactor the `start_game` system to handle user inputs from both keyboard and gamepad:
    - Use the `ButtonInput<KeyCode>` resource to detect keyboard inputs.
    - Use a query on the `Gamepad` component to handle gamepad inputs.
- Add a system `despawn_menu` to remove menu entities when exiting the `GameState::Menu` state.
- **Warning:** The application might crash if a camera is not available during state transitions.
- Create a system in `main.rs` to spawn a `Camera2d` when entering the `GameState::Menu` state.

REFACTOR SYSTEMS IN MAIN.RS TO ROTATE THE VESSEL

- Rename `my_first_system` to `setup_vessel`, remove all `println!` calls, and attach the `Player` component to the entity with the sprite component.
- Rename `my_second_system` to `rotate_vessel`, remove all `println!` calls, and keep only `let mut player = players.single_mut();` to perform the query.
- Modify the query to retrieve the entity that has both the `Player` and `Transform` components.
- Rotate the vessel around the Z-axis based on user inputs using `player.rotate_z(PI / 24.0);`.
- **Practical Exercise:**

Branch	Files	Estimated Time
09-inputs	src/main.rs src/states.rs	15 minutes

DEBUG CAMERA

REFACTOR CAMERA INTO A PLUGIN AND ADD A DEBUG CAMERA

- Refactor the camera systems into a `CameraPlugin` for better modularity.
- Add a system `debug_camera` to handle camera adjustments in debug mode.
- The `debug_camera` system should modify the `OrthographicProjection` scale based on user inputs (`W` to zoom in and `X` to zoom out).
- **Practical Exercise:**

Branch	Files	Estimated Time
10-debug_camera	src/main.rs src/camera.rs	5 minutes

DISPLAY ASTEROIDS

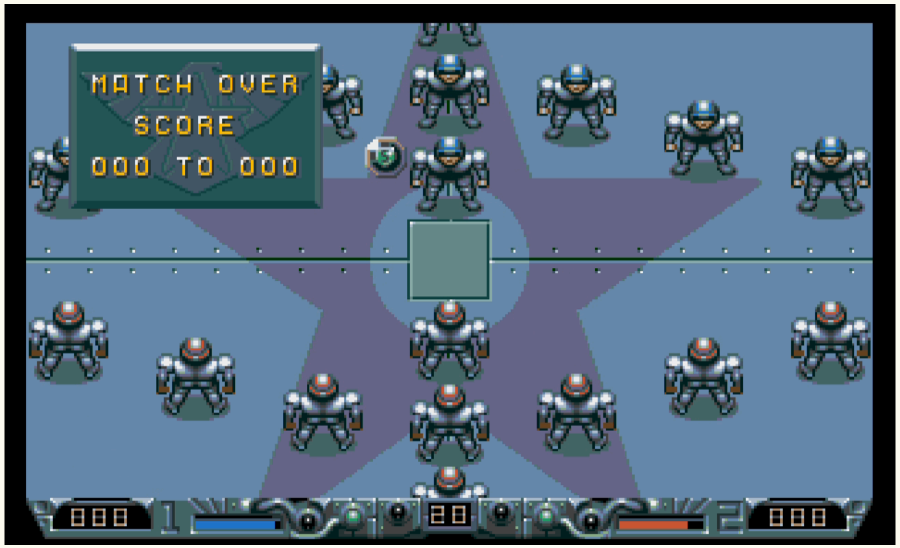
CREATE AN ASTEROID PLUGIN THAT SPAWN ASTEROIDS

- Create an `AsteroidPlugin` to improve modularity.
- Define an `Asteroid` structure with the fields `position`, `speed`, and `size`.
- Create an `AsteroidSize` structure and implement two methods:
  - `size()`: Returns the size of the asteroid.
  - `sprite()`: Returns the file name of the asteroid\u2019s sprite.
- Implement two systems:
  - `setup_asteroids`: Creates the field of asteroids.
  - `despawn_asteroids`: Removes asteroids when necessary.
- The `setup_asteroids` system should spawn 200 asteroids at random sizes and positions, ensuring no overlaps.
- **Practical Exercise:**

Branch	Files	Estimated Time
11-asteroids	src/camera.rs	10 minutes

MOVEMENTS

ASTEROIDS AND VESSEL MOVEMENTS



We could use our own physics "engine" to handle movement and collisions, as the required physics are relatively simple.

Alternatively, we could use Rapier for physics. This would give us an introduction to a physics engine and save development effort.

MOVING ASTEROIDS

- Initialize the `RapierPhysicsPlugin` and the `RapierDebugRenderPlugin` in the main function.
- Define the pixel per meter ratio used by the `RapierPhysicsPlugin`.
- Modify the `Asteroid` struct to include an additional field: `rot_speed: f32`.
- Add the following components from Rapier to the entities spawning asteroids to handle physics:
    - `RigidBody::Dynamic` - Declares the entity as a dynamic kinematic object.
    - `Collider::ball` - Defines a collider. You can use the `radius` method from the `AsteroidSize` implementation.
    - `GravityScale` - Sets gravity to `0.0` so that asteroids won't fall.
    - `Velocity` - Defines an initial linear and angular velocity for the asteroids.
- Implement wrapping for the asteroids if they move farther than four screen widths or heights in any direction.
- **Practical Exercise:**

Branch	Files	Estimated Time
12-movements	src/main.rs src/asteroids.rs	7 minutes

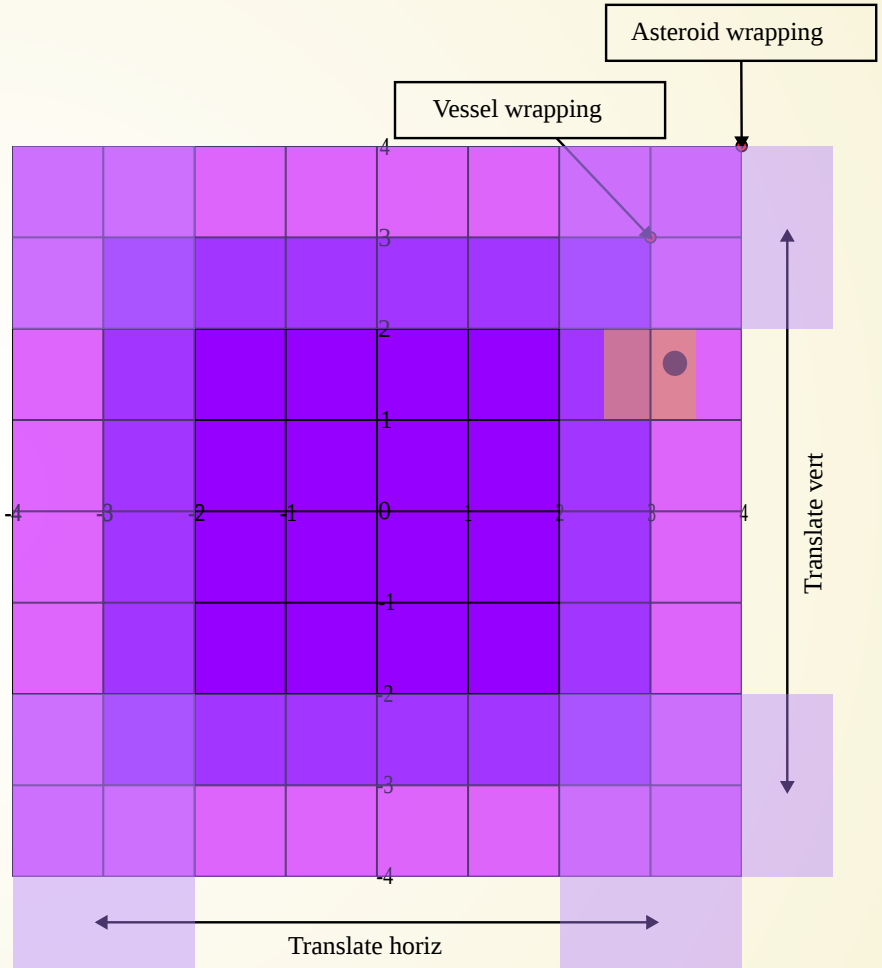
MOVING VESSEL

- Refactor the code and add a `VesselPlugin` for better modularity.
- Define a couple of constants to set the vessel's dimensions (e.g., `VESSEL_WIDTH`, `VESSEL_HEIGHT`).
- Add the following components from Rapier to the entity spawning the vessel to handle physics:
    - `RigidBody::Dynamic` - Declares the entity as a dynamic kinematic object.
    - `Collider::ball` - Defines a circular collider using `VESSEL_WIDTH / 4.0`.
    - `GravityScale` - Sets gravity to `0.0`, ensuring the vessel doesn't fall.
    - `ExternalImpulse` - Allows for velocity changes; initialize with `ExternalImpulse::default()`.
- Add a `move_vessel` system to handle user input. This system should generate an impulse in the direction the vessel is facing. Run the code to ensure the vessel is moving properly.
- Add a `stick_camera_on_vessel` system to the `CameraPlugin` that will move (stick) the camera to the vessel position.
- **Practical Exercise:**

Branch	Files	Estimated Time
12-movements	src/main.rs src/vessel.rs src/camera.rs	10 minutes

WRAPPING VESSEL

- The goal is to give the player the impression of infinite space, even though the playable area is limited to 6 screens in size.
- Mechanism
  - When the player moves beyond 3 screens in any direction
  - The vessel is translated back by 6 screens in the opposite direction.
  - Visible objects are translated and swapped to maintain their positions relative to the vessel.
- **Visible Objects:** Defined as objects within 1 screen of the vessel, with an additional 0.5 screen margin for safety.
- **Example:**
  - If the player moves more than 3 screens to the right:
  - The vessel is translated to -3 screens, and objects between screens 2 and 4 are translated to screens -4 and -2.
  - Objects between screens -4 and -2 are translated to screens 2 and 6.
- Implement a `wrap_vessel` system to handle the above mechanism.



- **Practical Exercise:**

Branch	Files	Estimated Time
12-movements	src/vessel.rs	10 minutes

EVENTS

COLLISIONS AND EXPLOSIONS

COLLISION EVENTS AND TIMERS

- **Concept**: Events are critical as they allow communication between systems. In this example, we will use a pre-existing event provided by Rapier, though custom events can also be created.
- Add the following components to the Player entity:
  - `ActiveEvents::COLLISION_EVENTS` - Enables collision event tracking.
  - `Visibility::Visible` - Allows control over the player's visibility.
  - `Velocity::default()` - Initializes default velocity (to be used later).
- Create a `vessel_collision` system to handle collision events:
  - In the initial implementation, log collision events using `debug!()`.
  - Later, change the Player's visibility to hidden and spawn an explosion animation.
- Explosion animations can be created by spawning the following components:
  - An `Explosion` component to identify the entity.
  - An `AnimationTimer` component that wraps a `Timer` component.
  - A `Sprite` component using a `TextureAlias` to display the animation.
- **Practical Exercise:**

Branch	Files	Estimated Time
13-explosions	src/vessel.rs	7 minutes

ANIMATIONS

- Create an `animate_player_explosion` system to update the texture atlas index of the `Sprite` component.
- The animation should loop by cycling through the index values from 0 to 8.
- The animation should be displayed on top of the vessel, which is hidden.
- **Practical Exercise:**

Branch	Files	Estimated Time
13-explosions	src/vessel.rs	7 minutes

OBSERVERS

MANAGE RESTART, GAMEOVER AND EXIT GAME

PROPERLY MANAGE RESTART WITH AN OBSERVER

- Managing the restart within the `animate_player_explosion` system would create unwanted coupling between systems.
- Events or observers are a better solution to ensure systems maintain single responsibility.
- Add a `Restart` event.
- Trigger the `Restart` event at the end of the explosion animation.
- Add an observer system (a system with a `Trigger` as its first parameter) to handle the restart:
    - Set the player's visibility to visible.
    - Reset the player's velocity to default.
    - Translate the vessel back to the origin and rotate it to angle 0.
    - Decrease the player's number of lives. If no lives remain, transition the `AppState` to `Gameover`.
- Add the observer system to the app.
- Add a `despawn_vessel` system to despawn the vessel when `Ingame` state is exited.
- **Practical Exercise:**

Branch	Files	Estimated Time
14-observers	src/vessel.rs	7 minutes

MANAGE GAMEOVER AND EXIT GAME

- Create a `display_gameover` system by duplicating the `display_menu` system.
- Add the `display_gameover` system to run when the `Gameover` state is entered.
- Rename the `start_game` system to `manage_inputs`, and update it to handle input controls in the `Gameover` state.
- Trigger the `AppExit::Success` event if the player presses the Escape key to exit the game.
- Update the app to execute the `manage_inputs` system in the `Gameover` state.
- Note: A potential issue could arise with the camera transitioning back to the `Menu` state. Handle this case carefully.
- **Bonus:** Add an `exit_to_menu` system in `vessel.rs` to allow returning to the `Menu`.
- **Practical Exercise:**

Branch	Files	Estimated Time
14-observers	src/state.rs src/vessel.rs src/camera.rs	7 minutes

VESSEL LASERS

LASERS

- Create a `display_gameover` system by duplicating the `display_menu` system.
- Add the `display_gameover` system to run when the `Gameover` state is entered.
- Rename the `start_game` system to `manage_inputs`, and update it to handle input controls in the `Gameover` state.
- Trigger the `AppExit::Success` event if the player presses the Escape key to exit the game.
- Update the app to execute the `manage_inputs` system in the `Gameover` state.
- Note: A potential issue could arise with the camera transitioning back to the `Menu` state. Handle this case carefully.
- **Bonus:** Add an `exit_to_menu` system in `vessel.rs` to allow returning to the `Menu`.
- **Practical Exercise:**

Branch	Files	Estimated Time
15-lasers	src/state.rs src/vessel.rs src/camera.rs	7 minutes

COMPILATION AND EXPORTATION

BUILDING AND EXPORTING

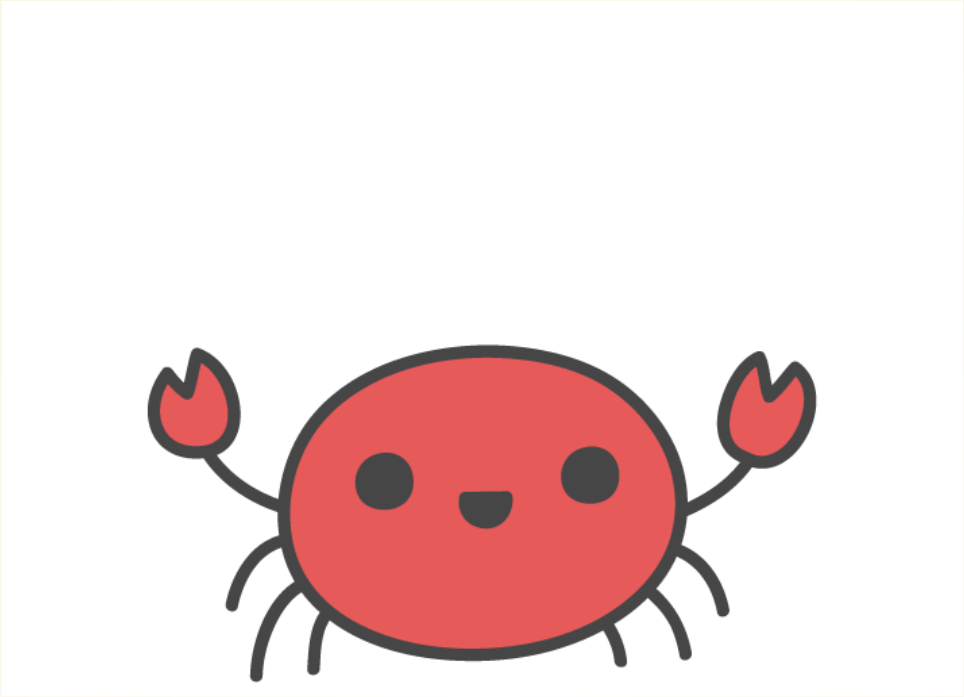
- **Native Compilation**: Test the mockup locally.
- **WebAssembly Compilation**: Deploy on a simple webpage.

PRACTICAL EXERCISE

Integrate the game mockup into a webpage and test the WebAssembly version.

ACHIEVEMENTS AND Q&A

Celebrate our accomplishments and address questions or feedback.



- Bastion Sevajol <bastien@sevajol.fr>
- René Ribaud <rene.ribaud@gmail.com>