Game Engine: Полная документация по использованию

Версия документа: 1.0 Актуально для кода в текущем репозитории (edition = 2024, wgpu = 27.0.1, winit = 0.30.12, egui = 0.33.3).

Содержание

1. О проекте и текущие возможности

Движок сейчас ориентирован на 2D-сцены с текстурами и новелльным UI:

• отрисовка спрайтов через wgpu;

• ортографическая камера;

• слои (Background, Character, Ui) + z_index;

• создание/обновление объектов сцены через сценарные команды;

• сценарные паузы (wait(seconds)) и применение изменений (apply(...));

• диалоговые окна через egui, отрисованные поверх 3D/2D кадра;

• обработка input-событий (клавиатура, мышь, колесо) + action map;

• пропуск ожидания в сценарии через Space/Enter, выход через Escape.

Что важно понимать:

• это не ECS и не полноценный редактор, а компактное ядро рендера + сценарий;

• объекты идентифицируются либо явным id, либо авто-ключом;

• сценарий в текущем виде линейный (очередь команд), без ветвлений.

2. Быстрый старт

2.1. Запуск

```
cargo run
```

2.2. Где задаётся сцена

Главный файл для конфигурации контента:

• src/scene_objects.rs

Там функция:

• read_initial_scene_script() -> Vec<SceneCommand>

Именно она формирует стартовую последовательность команд:

• spawn(...) — создать/показать объект;

• wait(...) — пауза в секундах;

• apply(...) — обновить существующий объект (например, скрыть).

3. Архитектура движка

3.1. Ключевые модули

• src/main.rs — цикл приложения, обработка событий, кадр.

• src/state.rs — инициализация wgpu (Instance, Surface, Device, Queue, SurfaceConfiguration).

• src/tex.rs — рендерер спрайтов, загрузка текстур, пайплайны, сортировка по слоям.

• src/game_object.rs — структуры GameObject2D, DialogueBoxObject, SceneObject.

• src/scene_script.rs — сценарные команды и таймлайн их выполнения.

• src/scene_objects.rs — декларация начального сценария сцены.

• src/dialogue_ui.rs — отрисовка диалоговых окон через egui.

• src/input.rs — состояние ввода и action-бинды.

• src/shader.wgsl — вершинный/фрагментный шейдер для спрайтов.

3.2. Поток кадра

• winit присылает WindowEvent.

• InputState обновляет флаги и очередь input-событий.

• SceneTimeline::update(dt, ...) исполняет команды сценария.

• Tex::render(...) рисует спрайты.

• DialogueUi::render(...) рисует UI-слой поверх кадра.

• frame.present() показывает кадр.

4. Жизненный цикл приложения (`main.rs`)

4.1. `resumed(...)`

При старте:

• создаётся окно;

• инициализируется State;

• создаются Tex и DialogueUi;

• создаётся SceneTimeline из read_initial_scene_script();

• сценарий выполняется один раз с dt = 0.0 для мгновенного применения стартовых spawn.

4.2. `window_event(...)`

Обрабатываются:

• ввод (self.input.on_window_event(&event));

• события для egui (dialogue_ui.on_window_event(...));

• CloseRequested;

• RedrawRequested;

• Resized.

4.3. Кадровая логика (`RedrawRequested`)

Внутри кадра:

• action Exit => закрытие приложения;

• action SkipWait => scene_timeline.skip_wait();

• вычисление dt через Instant;

• обновление сценария;

• рендер сцены и UI;

• input.end_frame() для сброса just_pressed/just_released.

5. Система рендера 2D-объектов (`tex.rs`)

5.1. Что хранит `Tex`

• GPU буферы вершин/индексов;

• layout для текстурных и uniform bind group;

• основной pipeline + опциональный wireframe pipeline;

• матрицу view_proj;

• список RenderObject;

• object_lookup: HashMap<String, usize> для быстрого поиска объектов по ключу.

5.2. Координаты и камера

Используется ортографическая проекция:

• X диапазон зависит от аспект-рейшио;

• Y примерно в диапазоне [-2, 2];

• камера смотрит вдоль -Z из точки (0,0,5).

Позиция и масштаб объекта задаются в world-space через:

• position: [f32; 2]

• scale: [f32; 2]

5.3. Создание и обновление объектов

Основные методы:

• create_game_object_from_definition(...) — загрузка текстуры и добавление нового объекта;

• apply_game_object_from_definition(...) — upsert: обновить существующий или создать новый;

• update_existing_object(...) — меняет transform/hidden и при необходимости перезагружает текстуру.

5.4. Сортировка

Сортировка объектов идёт по ключу:

• layer.order()

• z_index

• внутренний order (порядок добавления)

Это означает: для одинакового слоя и z_index раньше добавленный объект рисуется раньше.

5.5. Resize

resize(...) пересчитывает view_proj и обновляет матрицы всех объектов в uniform-буферах.

6. Система объектов сцены (`game_object.rs`)

6.1. `GameObject2D`

Поля:

• id: Option<String> — опциональный стабильный ключ;

• position, scale;

• texture_path;

• layer, z_index;

• hidden.

Важные методы:

• new(...)

• with_hidden(bool)

• with_id("...")

• scene_key()

6.2. `DialogueBoxObject`

Поля:

• id: Option<String>

• speaker

• text

• hidden

Важные методы:

• new(text) (по умолчанию speaker = "Lena")

• with_speaker(...)

• with_hidden(...)

• with_id(...)

• scene_key()

6.3. `SceneObject`

Единый enum для сценария:

• SceneObject::Sprite(GameObject2D)

• SceneObject::Dialogue(DialogueBoxObject)

Это позволяет одной очередью команд управлять и спрайтами, и UI-диалогами.

7. Сценарная система (`scene_script.rs`)

7.1. Команды

• spawn(object)

• apply(object)

• wait(seconds)

Тип команды:

• SceneCommand::Spawn(SceneObject)

• SceneCommand::Apply(SceneObject)

• SceneCommand::Wait(f32)

7.2. `SceneTimeline`

Внутри:

• pending: VecDeque<SceneCommand>

• wait_remaining: f32

Методы:

• new(commands)

• update(dt, device, queue, tex, dialogue_ui)

• skip_wait()

• is_finished()

7.3. Поведение `update(...)`

• если активен wait, он уменьшается на dt;

• когда wait заканчивается, выполнение команд продолжается в этом же кадре;

• Spawn и Apply сейчас одинаково применяются как upsert (создать или обновить).

7.4. Зачем `skip_wait()`

Позволяет игроку/пользователю пропускать тайминги, что особенно полезно для VN-сцен.

8. Конфигурация начальной сцены (`scene_objects.rs`)

Здесь определяется всё содержимое на старте.

Пример текущего подхода:

```
let game_object = GameObject2D::new(
    [0.0, 0.0],
    [1.0, 1.0],
    "src/image.jpg",
    RenderLayer::Character,
    5,
).with_hidden(false);

vec![
    spawn(game_object.clone()),
    spawn(DialogueBoxObject::new("But there are only two baskets.")),
    wait(5.0),
    apply(game_object.with_hidden(true)),
]
```

Практические рекомендации

• если объект будет обновляться много раз, задавайте with_id("hero"), чтобы ключ был стабильным;

• храните базовый объект в переменной и меняйте через builder (with_hidden, with_speaker и т.д.);

• группируйте команды логическими блоками (фон, персонажи, UI, паузы).

9. UI диалогов в стиле новеллы (`dialogue_ui.rs`)

9.1. Что делает модуль

• принимает DialogueBoxObject из таймлайна;

• хранит их список + lookup по scene_key;

• в каждом кадре строит egui-команды и рисует их поверх сцены.

9.2. Визуальные параметры

• положение: нижняя часть экрана;

• ширина: ~86% ширины viewport;

• высота: ~22% высоты viewport (минимум 120 px);

• фон: тёмный полупрозрачный;

• рамка: зеленоватая;

• speaker: фиолетовый;

• text: светло-жёлтый.

9.3. Обновление диалогов

apply_dialogue_object(...):

• если найден объект с тем же scene_key, обновляет его;

• если нет, добавляет как новый.

Это позволяет делать:

• показ/скрытие диалога;

• замену текста;

• смену speaker.

10. Система ввода (`input.rs`)

10.1. `InputState`

Отслеживает:

• зажатые клавиши (pressed_keys);

• нажатые в текущем кадре (just_pressed_keys);

• отпущенные в текущем кадре (just_released_keys);

• аналогично для кнопок мыши;

• позицию курсора;

• очередь InputEvent.

10.2. `InputEvent`

В очереди доступны события:

• KeyPressed(KeyCode) / KeyReleased(KeyCode)

• MousePressed(MouseButton) / MouseReleased(MouseButton)

• CursorMoved { x, y }

• MouseWheel { delta_y }

10.3. Action map

ActionMap по умолчанию:

• Action::SkipWait => Space, Enter

• Action::Exit => Escape

Проверка:

```
if self.action_map.just_pressed(Action::SkipWait, &self.input) {
    scene_timeline.skip_wait();
}
```

10.4. Ключевой момент кадра

В конце кадра обязательно:

```
self.input.end_frame();
```

Иначе just_pressed останется активным больше одного кадра.

11. Слои рендера и порядок отрисовки

11.1. RenderLayer

• Background (order = 0)

• Character (order = 1)

• Ui (order = 2)

11.2. Итоговый порядок

Сперва слой, затем z_index, затем порядок добавления.

11.3. Практика

• фон: Background + низкий z_index;

• персонажи: Character;

• спрайты интерфейса (не egui): Ui.

12. Практические рецепты

12.1. Скрыть спрайт через 3 секунды

```
let tree = GameObject2D::new([0.2, -0.1], [0.8, 0.8], "src/happy_tree.png", RenderLayer::Character
    .with_id("tree_1")
    .with_hidden(false);
```

```
vec![
    spawn(tree.clone()),
    wait(3.0),
    apply(tree.with_hidden(true)),
]
```

12.2. Поменять текст диалога

```
let line = DialogueBoxObject::new("Первый текст").with_id("line_main");

vec![
    spawn(line.clone()),
    wait(2.0),
    apply(
        DialogueBoxObject::new("Второй текст")
            .with_id("line_main")
            .with_speaker("Lena")
    ),
]
```

12.3. Пропускать тайминги пользователем

Уже реализовано по умолчанию через Space/Enter.

13. Ошибки, диагностика и ограничения

13.1. Частые проблемы

• failed to load texture '...': ...

Причина: неверный путь к файлу текстуры.

• Объект не обновляется через apply(...)

Причина: другой scene_key (например, отличился text у диалога или параметры у спрайта).

• Диалог не виден

Проверь hidden, и что в кадре вызывается dialogue_ui.render(...).

13.2. Ограничения текущей версии

• нет аудио;

• нет системы анимаций/таймлайна с easing;

• нет сериализации/сохранений;

• нет сложной UI-навигации;

• сценарий линейный, без веток и условий.

13.3. Производительность

Сейчас нормально для небольших 2D-сцен. Для роста проекта стоит добавить:

• батчинг текстур/атласы;

• кэширование одинаковых текстур;

• более формальную scene graph / ECS модель.

14. Расширение движка: куда добавлять новые фичи

14.1. Новые типы команд

Добавляйте в scene_script.rs:

• новые варианты SceneCommand (например, PlaySfx, MoveTo, FadeDialog);

• обработку в SceneTimeline::update.

14.2. Новые input-actions

Добавляйте в input.rs:

• enum Action;

• поля в ActionMap;

• логику в just_pressed(...).

14.3. Система персонажей

Рекомендуемый путь:

• стабильные with_id(...) для ключевых спрайтов;

• команды apply(...) для смены позы/текстуры;

• отдельный helper-модуль для фабрик объектов персонажей.

14.4. Ветвление сценария

Можно расширить SceneTimeline:

• добавить пользовательское состояние (flags, variables);

• добавить команды Jump(label) и If(condition, label).

15. Полный пример сценария

```
use crate::{
    game_object::{DialogueBoxObject, GameObject2D, RenderLayer},
    scene_script::{apply, spawn, wait, SceneCommand},
};

pub fn read_initial_scene_script() -> Vec<SceneCommand> {
    let bg = GameObject2D::new([0.0, 0.0], [2.0, 2.0], "src/happy_tree.png", RenderLayer::Backgrou
        .with_id("bg");

    let hero = GameObject2D::new([0.8, -0.2], [0.9, 0.9], "src/image.jpg", RenderLayer::Character,
        .with_id("hero")
        .with_hidden(false);

    let line = DialogueBoxObject::new("Привет. Это тестовая сцена.")
        .with_id("line")
        .with_speaker("Lena");

    vec![
        spawn(bg),
        spawn(hero.clone()),
```

```
        spawn(line),
        wait(2.0),
        apply(
            DialogueBoxObject::new("Нажми Space, чтобы пропустить ожидания.")
                .with_id("line")
                .with_speaker("Lena")
        ),
        wait(5.0),
        apply(hero.with_hidden(true)),
        apply(
            DialogueBoxObject::new("Персонаж скрыт через apply + with_hidden(true)")
                .with_id("line")
                .with_speaker("Lena")
        ),
    ]
}
```

Заключение

Текущая версия движка уже покрывает основной pipeline для визуальной новеллы:

• сцена из 2D спрайтов;

• тайминг-сценарий;

• UI диалогов;

• input-actions для управления сценарием.

Если продолжать развитие, самые полезные следующие шаги:

• анимации (перемещение/прозрачность/масштаб с easing);

• ветвления сценария и переменные;

• аудио;

• сериализация сцен и внешние сценарные файлы.