1) Физический маятник (SFML)

Численное моделирование + визуализация, работа с dt и устойчивостью.

Требования к функционалу

- Окно SFML, визуализация маятника (стержень + груз), ось подвеса.
- Выбор метода интегрирования: Euler (для сравнения) и RK4 (основной).
- Параметры: длина, масса, g, коэффициент демпфирования; ввод с клавиатуры/меню.
- Пауза/старт, сброс, изменение скорости симуляции, отображение FPS.
- Графики θ(t) и ω(t) или лог в файл CSV (минимум одно).

Компоненты и логика

- Model: состояние (θ, ω), параметры (L, m, g, damping).
- Integrator: step(state, dt) для Euler и RK4.
- Renderer (SFML): преобразование угла в координаты (x,y), рисование.
- Controller: обработка ввода, управление dt, pause, reset.
- DataLogger: CSV или кольцевой буфер для графика.

2) Орбита спутника Земли (SFML)

Двухтельная задача, эллиптические орбиты, скорость/энергия/перигей.

Требования к функционалу

- 2D-визуализация Земли и траектории спутника, масштабирование/камера.
- Модель гравитации: $F = GMm/r^2$ (использовать $\mu=GM$ и m сократить).
- Стартовые условия: позиция и скорость (вектор), пресеты орбит.
- Пауза/старт, шаг симуляции, отображение параметров: r, v, энергия.
- Отрисовка следа траектории (polyline) и отметок перигея/апогея (опция).

Компоненты и логика

- Physics: state (pos, vel), accel(pos)= -μ * pos / |pos|^3.
- Integrator: semi-implicit Euler или Velocity Verlet (рекоменд.), сравнить с RK4 (опц.).
- OrbitMetrics: энергия, момент импульса, оценка a/e, перигей/апогей.
- Renderer: world→screen transform, zoom/pan, trail buffer.
- UI/Controller: пресеты, ручной ввод v, reset.

3) Тетрис (SFML)

Классический gameplay: фигуры, вращение, линии, очки, уровни.

Требования к функционалу

- Поле 10×20, 7 фигур (I,O,T,S,Z,J,L), генератор очереди (bag 7, опц.).
- Движение: влево/вправо/вниз, hard drop, удержание (hold, опц.).
- Вращение с проверкой коллизий; минимум простая wall-kick логика.
- Очистка линий, подсчёт очков, ускорение с уровнем, Game Over.
- Отображение next-панели и текущего счета.

Компоненты и логика

- Board: сетка int[H][W], методы collide(), lockPiece(), clearLines().
- Piece: форма 4×4, позиция, rotation; методы rotate(), cells().
- Game: state machine (Playing/Paused/GameOver), таймер падения.
- Input: обработка повторов (DAS/ARR упрощённо, опц.).
- Renderer (SFML): отрисовка клеток, UI, next/hold.

4) Простая база данных (файлы + JSON)

Мини-СУБД: таблицы/записи, CRUD, индексация, сохранение на диск.

Требования к функционалу

- Хранение таблицы как набора JSON-документов (JSON Lines) или одного файла JSON.
- Команды CLI: create-table, insert, select (фильтр), update, delete.
- Схема полей (тип string/int/double/bool) и валидация при insert/update.
- Индекс по одному полю (например, id) для быстрого поиска.
- Журналирование (лог операций) или бэкап (опция).

Компоненты и логика

- Storage: чтение/запись файла, атомарная запись (temp+rename).
- Parser/Serializer: JSON (nlohmann/json) и преобразование типов.
- Table: schema, vector/iterator records, CRUD операции.
- Index: unordered_map<key, offset/rowid> (для JSONL) или map rowid.
- CLI: разбор аргументов, форматирование вывода.

5) Autograd на C++

Автодифференцирование для тензоров 1D/2D: граф вычислений + backprop.

Требования к функционалу

- Класс Tensor (данные + grad + requires_grad), операции: +, -, *, /, matmul (опц.).
- Поддержка нескольких операций: sum, mean, relu, sigmoid (минимум две).
- Построение графа и backprop: tensor.backward() вычисляет градиенты.
- Тесты градиентов на простых выражениях (проверка численным градиентом, опц.).
- Небольшой пример обучения: линейная регрессия или логистическая (опц.).

Компоненты и логика

- Tensor: хранит value (vector<double>) и grad, ссылку на Node/Op.
- Node/Op: тип операции, ссылки на родителей, функция backward().
- Engine: topological sort графа, обратный проход, аккумулирование grads.
- Ops: реализовать forward + локальные производные.
- Examples/Tests: минимальные кейсы и сравнение с ожидаемыми grad.

6) 2D Space Shooter (SFML)

Игровая архитектура: сущности, коллизии, стрельба, AI, уровни.

Требования к функционалу

- Игрок: управление, стрельба, здоровье/щит, смерть и перезапуск.
- Враги: спавн волнами, простое поведение (движение + стрельба/таран).
- Коллизии: пули↔корабли, корабль↔корабль; эффекты попаданий.
- Система очков, UI (HP, score), экраны pause/game over.
- Звук/частицы — опционально, но приветствуется.

Компоненты и логика

- Entity: позиция/скорость/радиус; Player/Enemy/Bullet.
- Systems: Movement, Collision, Spawn, Combat, UI.
- GameState: Playing/Paused/GameOver + управление сценой.
- ResourceManager: загрузка текстур/шрифтов/звуков.
- Renderer (SFML): слой фон/объекты/UI.

7) Ray Tracing (CPU)

Рендер сцены: лучи, сферы/плоскости, освещение, отражения (опц.).

Требования к функционалу

- Рендер в изображение (PPM/PNG): камера, viewport, пиксельный цикл.
- Примитивы: минимум сфера + плоскость; материалы: diffuse + metal (опц.).
- Освещение: точечный источник и тени (shadow ray).
- Антиалиасинг (сэмплы на пиксель) и gamma correction.
- Сцена из конфигурации (JSON) — опционально.

Компоненты и логика

- Math: Vec3, Ray, операции, normalize, dot/cross.
- Geometry: intersect(ray) -> Hit{t, point, normal, material}.
- Materials: shade(hit, lights) и (опц.) отражение/рассеяние.
- Camera: генерация первичного луча по (x,y) с учетом FOV.
- ImageWriter: PPM (обяз.), PNG (опц. через stb_image_write).