TACHBIT · КУРС НА 30 ДНЕЙ · ЧАСТЬ 2

Как думает программа

Часть 2. Дни 15–30: как устроены игры и операционная система. 16 паттернов, по одному в день, с проверенным кодом на Tachbit.

Приложение
ждёт события

Игра
живёт 60 кадров в секунду

ОС
служит всем остальным

Версия от 25 ноября 2026 · Код проверен компилятором Tachbit 1.3

Большая картина

Приложение, игра и ОС выглядят совсем разными, но внутри у них один скелет. Есть **состояние** (что программа помнит), **события** (что его меняет), **цикл** (кто ждёт события) и **слои** (кто за что отвечает). Разница только в том, как крутится цикл.

Приложение

Спит, пока ничего не происходит. Клик или сообщение будит его, оно меняет данные, перерисовывает окно и снова засыпает.

Игра

Не спит никогда. 60 раз в секунду: сколько прошло времени, что нажато, сдвинуть мир, нарисовать кадр заново.

Операционная система

Служит другим программам: делит между ними процессор, память, диск и экран, следит, чтобы никто не мешал другому.

Цикл приложения

ждать событие → разобрать → изменить состояние → перерисовать ↺

Цикл игры

dt → ввод → обновить мир → нарисовать кадр ↺ 60 раз в секунду

Цикл ядра ОС

прерывание → кто это: таймер, клавиша, диск → разбудить нужную службу → выбрать следующую задачу ↺

План на 30 дней

Дни 1–7	**Основа логики** Состояние, решения, циклы, модели, функции, ошибки. Без этого не работает ни одна программа.
Дни 8–14	**Как устроены приложения** Справочники, сохранение, слои, цикл событий, модель и вид, автоматы, отмена.
Дни 15–22	**Как устроены игры** Игровой цикл, физика, столкновения, сущности, экраны, плавность, события, ИИ.
Дни 23–30	**Как устроена система** Службы, память, параллельность, очереди, буферы, планировщик, права, общий скелет.

Как заниматься

Около часа в день. Прочитайте объяснение и наберите код руками: не копируйте его, так логика запоминается лучше. Запустите `tb run dayNN.tb` и сравните вывод с тем, что в книге. Потом сделайте задание. Если задание не получилось, это нормально: вернитесь к нему через день.

Готовые файлы примеров лежат рядом с этой книгой в архиве `tachbit-kurs-kod.zip`. Курс разделён на две книги: в первой части дни 1–14 (логика и приложения), во второй дни 15–30 (игры и система).

15 Игровой цикл

Цель: Понять, чем игра отличается от приложения: она живёт, даже когда вы ничего не нажимаете.

Приложение спит, пока нет событий. Игра крутится постоянно, обычно 60 раз в секунду. Каждый круг: узнать, сколько прошло времени (`dt`), прочитать ввод, обновить мир, нарисовать всё заново.

Важная деталь — `dt` . Если двигать героя «на 3 точки за кадр», на быстром компьютере он побежит быстрее. Если «200 точек в секунду × dt», скорость одинакова везде.

ПРИЛОЖЕНИЕ
Видеоплеер и анимации интерфейса тоже живут в таком цикле, пока что-то движется.

ИГРА
Каждая игра: футбол, гонки, платформер. В нашем `examples/apps/football` ровно этот цикл.

ОС
Оболочка «Капля» рисует окна в цикле по кадрам; таймер ядра тикает так же, только для задач.

Паттерн «Игровой цикл». Каждый кадр: dt → ввод → обновить → нарисовать. Скорость всегда × dt.

КОД: DAY15.TB

```
// день 15: игровой цикл — ввод, обновить, нарисовать
func main()
    try window_open(640, 360, "Игровой цикл")
    set_fps(60)
    x = 320.0
    speed = 200.0                       // точек в секунду
    repeat while window_alive()
        dt = frame_ms() / 1000          // 1) сколько прошло
        if key_down("left")             // 2) ввод
            x -= speed * dt
        if key_down("right")
            x += speed * dt
        x = max(20.0, min(620.0, x))    // 3) обновить и не выпустить за край
        clear_window(rgb(20, 24, 36))   // 4) нарисовать всё заново
        draw_circle(round(x), 180, 20, rgb(255, 190, 60))
```

ЧТО БУДЕТ

```
Откроется окно. Запускайте на своём компьютере, где есть экран.
```

Задание. Добавьте движение вверх-вниз и второй шарик, который медленно едет за первым.

16 Позиция, скорость, ускорение

Неделя 3

Цель: Двигать предметы правдоподобно тремя строками.

Почти вся игровая физика стоит на трёх числах. **Позиция** — где предмет. **Скорость** — насколько позиция меняется за секунду. **Ускорение** — насколько меняется скорость (тяжесть, толчок, тормоз).

Порядок в кадре всегда один: ускорение меняет скорость, скорость меняет позицию. Прыжок — это просто большая скорость вверх, а тяжесть сама вернёт героя на землю.

ПРИЛОЖЕНИЕ
Плавная прокрутка списка по инерции после свайпа — это скорость с трением.

ИГРА
Прыжок, падение, полёт мяча, разгон машины, отдача от выстрела.

ОС
Курсор мыши с ускорением: быстрое движение руки уводит курсор дальше.

Паттерн «Интегратор». скорость += ускорение × dt; позиция += скорость × dt.

КОД: DAY16.TB

```
// день 16: позиция, скорость, ускорение
func main()
    y = 0.0          // высота мяча
    vy = 12.0        // скорость вверх
    g = -9.8         // тяжесть меняет скорость
    dt = 0.1
    t = 0.0
    repeat while y ⩾ 0.0
        vy += g * dt         // ускорение меняет скорость
        y += vy * dt         // скорость меняет позицию
        t += dt
    print("мяч упал через", round(t * 10) / 10.0, "с")
```

ВЫВОД

```
мяч упал через 2.4 с
```

Задание. Добавьте отскок: когда мяч касается земли, скорость меняет знак и теряет 30%. Сколько отскоков до остановки?

17 Столкновения

Цель: Проверять, коснулись ли два предмета.

Самая частая проверка в играх — пересекаются ли два прямоугольника. Их называют AABB (прямоугольники без поворота). Две коробки **не** пересекаются, если одна целиком левее, правее, выше или ниже другой. Во всех остальных случаях — пересекаются.

Сначала проверяйте грубо (коробки), потом точно, если нужно. Это экономит время, когда предметов сотни.

ПРИЛОЖЕНИЕ
Клик мышью по кнопке — та же проверка точки в прямоугольнике. Перетаскивание карточки в колонку — пересечение.

ИГРА
Герой и стена, пуля и враг, мяч и ворота, монета и игрок.

ОС
Оконный менеджер: какое окно под курсором, какие окна перекрыты и что перерисовать.

Паттерн «Пересечение коробок». Не пересекаются, если одна целиком с одной стороны. Иначе — касание.

КОД: DAY17.TB

```
// день 17: столкновения — две коробки пересекаются?
obj Box
    x: dec
    y: dec
    w: dec
    h: dec

func hit(a: Box, b: Box) gives yes: bool
    yes = a.x < b.x + b.w and b.x < a.x + a.w and a.y < b.y + b.h and b.y < a.y + a.h

func main()
    hero = Box(x: 10, y: 10, w: 20, h: 30)
    wall = Box(x: 25, y: 0, w: 10, h: 100)
    coin = Box(x: 80, y: 15, w: 8, h: 8)
    print("герой и стена:", hit(hero, wall))
    print("герой и монета:", hit(hero, coin))
```

ВЫВОД

```
герой и стена: true
герой и монета: false
```

Задание. Напишите `point_in(b: Box, x: dec, y: dec)` и проверьте «клик» по монете.

18 Много сущностей

Цель: Управлять сотней одинаковых объектов как одним целым.

В игре одновременно живут десятки врагов, пуль, частиц. Их держат в списке и каждый кадр делают одно и то же: обновить всех, убрать «мёртвых», добавить новых.

Удалять из списка, по которому сейчас идёте, опасно: можно пропустить элемент. Надёжнее собрать новый список живых и заменить старый.

ПРИЛОЖЕНИЕ
Список уведомлений: каждое живёт 5 секунд, потом исчезает.

ИГРА
Пули, враги, искры, капли дождя.

ОС
Таблица процессов: завершённые убирают, новые добавляют, остальным раздают время.

Паттерн «Список сущностей». Обновить всех → собрать живых в новый список → заменить.

КОД: DAY18.TB

```
// день 18: много сущностей — список, обновить всех, убрать мёртвых
obj Enemy
    x: dec
    hp: int

func main()
    enemies = list Enemy
    count i from 1 to 5
        enemies.add(Enemy(x: i * 10.0, hp: i))
    // выстрел по всем: каждый теряет 2 жизни
    each e in enemies
        e.hp -= 2
    // убрать мёртвых: собрать новый список живых
    alive = list Enemy
    each e in enemies
        if e.hp > 0
            alive.add(e)
    enemies = alive
    print("осталось врагов:", len(enemies))
```

ВЫВОД

```
осталось врагов: 3
```

Задание. Каждый «кадр» двигайте врагов на +5 по x и убирайте тех, кто ушёл дальше 40.

19 Экраны игры

Цель: Применить автомат из дня 13 ко всей игре.

Меню, игра, пауза, конец — это режимы целой программы. Внутри каждого свои правила: в паузе мир не обновляется, в меню стрелки двигают выбор, а не героя.

Это тот же конечный автомат, что и дверь. Один `match` на режим, и каждый экран занимается только своим.

ПРИЛОЖЕНИЕ
Мастер установки: шаг 1, 2, 3, готово. Вход: логин → код из SMS → главная.

ИГРА
Заставка → меню → уровень → пауза → проигрыш → меню.

ОС
Загрузка → вход → рабочий стол → блокировка экрана → выключение.

Паттерн «Автомат экранов». Режим программы — один enum. Каждый экран обрабатывает только свои события.

КОД: DAY19.TB

```
// день 19: экраны игры — тот же автомат, что и дверь
enum Screen
    menu
    play
    pause
    over

func main()
    s = Screen.menu
    lives = 1
    each key in ["enter", "p", "p", "удар", "enter"]
        match s
            menu
                if key == "enter"
                    s = Screen.play
            play
                if key == "p"
                    s = Screen.pause
                else if key == "удар"
                    lives -= 1
                    if lives == 0
                        s = Screen.over
            pause
                if key == "p"
                    s = Screen.play
            over
                if key == "enter"
                    s = Screen.menu
        print(key, "→", s)
```

ВЫВОД

```
enter → play
p → pause
p → play
удар → over
enter → menu
```

Задание. Добавьте экран `win`: если в игре нажали «гол», перейти в него; из него `enter` ведёт в меню.

20 Плавность

Цель: Делать движение мягким, а не резким.

Когда окно появляется мгновенно, это выглядит дёшево. Плавность — это движение от «было» к «стало» за время. Доля пути `t` идёт от 0 до 1, а `lerp` даёт точку между началом и концом.

Если `t` растёт ровно, движение механическое. Кривая `ease` делает его живым: быстро в начале и мягко в конце. Библиотека `lib/anim.tb` уже содержит эти функции.

ПРИЛОЖЕНИЕ
Выезжающие панели, кнопки, которые «пружинят», переходы между страницами.

ИГРА
Камера, которая догоняет героя; полоска жизни, которая плавно уменьшается.

ОС
Тема «Капля»: окна раскрываются и закрываются с мягкой кривой.

Паттерн «Плавный переход». Значение = lerp(было, стало, ease(t)). t идёт от 0 до 1 по времени.

КОД: DAY20.TB

```
// день 20: плавность — от «было» к «стало» за время
use anim

func main()
    from_x = 0.0
    to_x = 100.0
    count step from 0 to 5
        t = step / 5.0                          // 0 … 1 — доля пути
        a = anim.lerp(from_x, to_x, t)          // ровно
        b = anim.lerp(from_x, to_x, anim.ease("out", t))   // быстро, потом мягко
        print("t =", t, " ровно", round(a), " мягко", round(b))
```

ВЫВОД

```
t = 0  ровно 0  мягко 0
t = 0.2  ровно 20  мягко 49
t = 0.4  ровно 40  мягко 78
t = 0.6  ровно 60  мягко 94
t = 0.8  ровно 80  мягко 99
t = 1  ровно 100  мягко 100
```

Задание. Сравните кривые `"in"` и `"inout"` и выведите таблицу из 11 шагов (t = 0, 0.1 ... 1).

21 События и слушатели

Неделя 3

Цель: Связывать части программы, не заставляя их знать друг о друге.

Когда мяч забивают, нужно: добавить очко, сыграть звук, показать надпись, записать статистику. Если физика мяча будет вызывать всё это сама, она станет знать обо всём на свете.

Вместо этого физика кладёт **событие** «гол» в очередь. Звук, счёт и надписи читают очередь и реагируют на то, что им интересно. Новую реакцию можно добавить, не трогая физику.

ПРИЛОЖЕНИЕ
«Карточка перешла в Done» → уведомление, письмо клиенту, запись в журнал.

ИГРА
«Враг побеждён» → очки, звук, выпадение предмета, достижение.

ОС
«Флешка вставлена» → значок на столе, автоматическое сканирование, звук.

Паттерн «Очередь событий». Источник только сообщает. Слушатели сами решают, что делать.

КОД: DAY21.TB

```
// день 21: события — кто-то сообщил, другие отреагировали
obj Event
    kind: text
    power: int

func main()
    events = list Event                 // очередь событий кадра
    events.add(Event(kind: "удар", power: 7))
    events.add(Event(kind: "монета", power: 1))
    events.add(Event(kind: "удар", power: 2))
    score = 0
    each e in events                    // каждый слушатель смотрит своё
        if e.kind == "удар" and e.power > 5
            print("звук: громкий удар")
        if e.kind == "монета"
            score += 10
            print("звук: дзынь")
    print("очки:", score)
```

ВЫВОД

```
звук: громкий удар
звук: дзынь
очки: 10
```

Задание. Добавьте событие «уровень» и слушателя, который выводит «новый уровень!» каждые 20 очков.

22 Простой ИИ

Цель: Научить персонажа принимать решения.

Игровой «интеллект» обычно очень простой: посмотреть на мир, выбрать действие по правилам, выполнить. Правила идут от самого срочного к самому обычному — это тот же охранник из дня 2.

Для сложного поведения правила объединяют с автоматом: у врага есть режимы «патруль», «погоня», «бегство», а `decide` выбирает действие внутри режима.

ПРИЛОЖЕНИЕ
Подсказки: «давно не отвечали клиенту → напомнить». Сортировка писем по правилам.

ИГРА
Вратарь следит за мячом, бот в футболе обходит мяч и бьёт в дальний угол.

ОС
Планировщик решает, кому дать процессор: срочным — первыми, долгим — по очереди.

Паттерн «Смотри → реши → действуй». Правила от срочного к обычному. Первое подходящее и выигрывает.

КОД: DAY22.TB

```
// день 22: простой ИИ — смотри, реши, действуй
func decide(my_x: dec, ball_x: dec, dist_goal: dec) gives act: text
    if abs(ball_x - my_x) > 1.0
        act = "бежать к мячу"           // далеко — догнать
    else if dist_goal < 10.0
        act = "бить по воротам"         // близко к воротам — бить
    else
        act = "вести мяч"               // иначе вести дальше

func main()
    print(decide(0.0, 5.0, 20.0))
    print(decide(4.5, 5.0, 20.0))
    print(decide(4.5, 5.0, 6.0))
```

ВЫВОД

```
бежать к мячу
вести мяч
бить по воротам
```

Задание. Добавьте правило «если у соперника мяч и он ближе 3, отбирать». Где его поставить в списке?

23 Службы и сообщения

Неделя 4

Цель: Понять, как устроена ОС Omnium: части общаются письмами, а не общими переменными.

Когда две программы меняют одну переменную одновременно, возникает путаница. В Omnium части системы — это **службы**. У каждой свои данные, и никто их не трогает. Чтобы что-то попросить, службе посылают сообщение по каналу и ждут ответ.

В Tachbit для этого есть `channel`: `send` кладёт сообщение, `receive` забирает, `each` перебирает, пока канал не закроют. `background` запускает службу рядом с основной программой.

ПРИЛОЖЕНИЕ
Фронт и бэкенд сайта: страница посылает запрос, сервер отвечает. Это та же служба.

ИГРА
Отдельный поток загружает уровни и посылает «готово» основному циклу.

ОС
В Omnium часы, клавиатура, консоль и диск — службы с каналами (omnium/clock.tb, keyboard.tb).

Паттерн «Служба». Свои данные у каждого. Просьба — сообщение в канал. Ответ — сообщение обратно.

КОД: DAY23.TB

```
// день 23: ОС — службы и сообщения вместо общих переменных
func clock_service(asks: channel text, answers: channel text)
    each q in asks                      // служба ждёт вопрос
        if q == "время"
            answers.send(date_now())
        else
            answers.send("не знаю: " + q)

func main()
    asks = channel text
    answers = channel text
    w = background clock_service(asks, answers)
    asks.send("время")
    print("ответ службы:", answers.receive())
    asks.close()
```

ВЫВОД

```
ответ службы: 2026-11-22 12:30:00
```

Задание. Научите службу отвечать на «привет» и на «сколько» (сколько вопросов она уже получила).

24 Память: кто владеет и когда отдать

Цель: Понять, откуда берётся память и почему её нужно возвращать.

Любые данные лежат в памяти. Обычно Tachbit сам решает, когда её освободить. Блок `temp` говорит прямо: всё, что тут создано, больше не нужно после блока.

В системном коде (драйверы, ядро) память берут вручную в блоке `danger`: `new_mem` — взять, `free_mem` — вернуть. Правило одно: у каждого куска есть хозяин, и хозяин обязан его вернуть ровно один раз. Забыли — утечка. Вернули дважды или пользуетесь после возврата — порча чужих данных.

ПРИЛОЖЕНИЕ
Открытые картинки и файлы закрывают, когда окно закрыли, иначе программа растёт часами.

ИГРА
Уровень загружают целиком в одну «зону» и выбрасывают целиком при переходе.

ОС
Omnium считает память каждой клетки и не даёт одной программе съесть всю (квота 16 МБ).

Паттерн «Владение». У каждого куска один хозяин. Взял — отдай ровно один раз.

КОД: DAY24.TB

```
// день 24: память — кто владеет и когда отдать
func main()
    temp                                    // всё, что создано тут, уйдёт в конце блока
        words = split("раз два три", " ")
        print("слов:", len(words))
    danger                                  // руками: взял — отдай
        buf = new_mem(64) else err
            print(err)
            return
        write32(buf, 42)
        print("в памяти:", read32(buf))
        free_mem(buf)
```

ВЫВОД

```
слов: 3
в памяти: 42
```

Задание. Возьмите 16 байт, запишите четыре числа `write32(buf + i*4, i)` и прочитайте их обратно.

25 Параллельно: каждый своё, общее под замком

Неделя 4

Цель: Разделить работу между ядрами процессора и не испортить общие данные.

У процессора несколько ядер. `background` запускает функцию на другом ядре, `.wait` ждёт её ответ. Лучше всего, когда каждый работник считает своё и ничего не делит.

Если общее всё же нужно, его объявляют `shared` и меняют только внутри `lock`. Замок пускает внутрь по одному. Tachbit проверяет это при сборке: менять `shared` без замка нельзя.

ПРИЛОЖЕНИЕ
Сжатие фото на всех ядрах, загрузка файлов, пока интерфейс остаётся живым.

ИГРА
Физика на одном ядре, звук на другом, загрузка уровня на третьем.

ОС
Omnium работает на нескольких ядрах сразу; общие данные ядра защищены замком.

Паттерн «Раздели и собери». Каждый работник — своя часть. Общее — shared и только под lock.

КОД: DAY25.TB

```
// день 25: параллельно — каждый своё, общее под замком
shared done = int 0

func work(a: int, b: int)
    sum = 0
    count i from a to b
        sum += i
    lock done
        done += 1                        // общее меняем только под замком
    return sum

func main()
    a = background work(1, 500000)
    b = background work(500001, 1000000)
    print("сумма:", a.wait + b.wait, " работ закончено:", done)
```

ВЫВОД

```
сумма: 500000500000  работ закончено: 2
```

Задание. Разделите сумму на 4 работника вместо двух и сравните время через `time_ms()` до и после.

26 Производитель и потребитель

Цель: Связать того, кто создаёт работу, с тем, кто её выполняет.

Один кладёт заказы, другой их готовит. Между ними канал — как окошко на кухне. Производитель не ждёт, пока блюдо будет готово, а потребитель не спрашивает каждую секунду «есть что-нибудь?» — он просто ждёт на канале.

Закрытие канала (`close`) — сигнал «заказов больше не будет». Цикл `each` по каналу сам закончится, когда всё прочитано.

ПРИЛОЖЕНИЕ
Очередь отправки писем, обработка загруженных файлов, печать документов.

ИГРА
Загрузчик ресурсов кладёт готовые текстуры, игра забирает их между кадрами.

ОС
Клавиатура кладёт нажатия, оболочка забирает. Диск кладёт прочитанные блоки, файловая система забирает.

Паттерн «Производитель / потребитель». Между ними канал. Закрыл канал — значит работы больше нет.

КОД: DAY26.TB

```
// день 26: производитель и потребитель
func cook(orders: channel int, ready: channel text)
    each n in orders
        ready.send("блюдо №" + n)
    ready.close()

func main()
    orders = channel int
    ready = channel text
    k = background cook(orders, ready)
    count i from 1 to 3
        orders.send(i)                    // производитель кладёт
    orders.close()
    each dish in ready                    // потребитель забирает
        print(dish)
```

ВЫВОД

```
блюдо №1
блюдо №2
блюдо №3
```

Задание. Запустите двух поваров на одном канале заказов и подпишите, кто какое блюдо приготовил.

27 Кольцевой буфер

Неделя 4

Цель: Хранить поток данных в памяти фиксированного размера.

Драйвер клавиатуры не может просить новую память на каждое нажатие — ему нужна заранее выделенная полка. **Кольцевой буфер** — это массив, где запись и чтение идут по кругу: дошли до конца — начинаем сначала.

Два указателя: `tail` — куда писать, `head` — откуда читать. Если полка полна, решают заранее: терять новое или перезаписывать старое.

ПРИЛОЖЕНИЕ
Последние 100 строк журнала, история чата в памяти, буфер записи звука.

ИГРА
Запись последних 5 секунд для повтора гола, сглаживание мыши по последним кадрам.

ОС
Буфер клавиатуры, сетевой карты, звуковой карты: железо пишет, драйвер читает.

Паттерн «Кольцо». Фиксированный массив, два указателя, всё по модулю длины.

КОД: DAY27.TB

```
// день 27: кольцевой буфер — как драйвер копит нажатия
obj Ring
    data: list int
    head: int      // откуда читать
    tail: int      // куда писать
    used: int

func put(r: Ring, x: int) gives ok: bool
    ok = r.used < len(r.data)
    if not ok
        return                          // полно: новое теряем, старое не портим
    r.data[r.tail] = x
    r.tail = (r.tail + 1) % len(r.data)
    r.used += 1

func take(r: Ring) gives x: int
    x = r.data[r.head]
    r.head = (r.head + 1) % len(r.data)
    r.used -= 1

func main()
    r = Ring(data: [0, 0, 0, 0])
    each key in [65, 66, 67, 68, 69]       // 5 нажатий в буфер на 4
        if not put(r, key)
            print("буфер полон, потеряно:", char(key))
    repeat while r.used > 0
        print("прочитал:", char(take(r)))
```

ВЫВОД

```
буфер полон, потеряно: E
прочитал: A
прочитал: B
прочитал: C
прочитал: D
```

Задание. Сделайте вариант, который при переполнении затирает самое старое значение.

28 Планировщик

Цель: Понять, как одна система даёт время многим программам.

Ядер мало, программ много. Планировщик даёт каждой задаче немного времени (**квант**), потом ставит её в конец очереди и берёт следующую. Так все двигаются вперёд, и никто не захватывает процессор навсегда. Это вариант «по кругу» (round-robin); настоящие планировщики добавляют приоритеты, но основа та же.

ПРИЛОЖЕНИЕ
Сервер раздаёт по чуть-чуть всем клиентам, чтобы один тяжёлый запрос не тормозил остальных.

ИГРА
Тяжёлые дела (поиск пути для 100 врагов) делят на куски: по 10 врагов за кадр.

ОС
Таймер ядра Omnium тикает и по очереди переключает задачи на всех ядрах.

Паттерн «По кругу». Очередь задач. Каждой — квант. Не закончила — в конец очереди.

КОД: DAY28.TB

```
// день 28: планировщик — всем по очереди понемногу
obj Task
    name: text
    left: int      // сколько ещё работы

func main()
    queue = list Task
    queue.add(Task(name: "музыка", left: 3))
    queue.add(Task(name: "браузер", left: 5))
    queue.add(Task(name: "часы", left: 1))
    slice = 2                              // квант: сколько даём за раз
    tick = 0
    repeat while len(queue) > 0
        t = queue[0]
        rest = list Task
        count i from 1 to len(queue) - 1
            rest.add(queue[i])
        work = min(slice, t.left)
        t.left -= work
        tick += work
        print("такт", tick, ":", t.name, "сделал", work)
        if t.left > 0
            rest.add(t)                    // не закончил — в конец очереди
        else
            print("   ", t.name, "готов")
        queue = rest
```

ВЫВОД

```
такт 2 : музыка сделал 2
такт 4 : браузер сделал 2
такт 5 : часы сделал 1
    часы готов
такт 6 : музыка сделал 1
    музыка готов
такт 8 : браузер сделал 2
такт 9 : браузер сделал 1
    браузер готов
```

Задание. Добавьте приоритет: задача с `urgent: bool` получает квант 4 вместо 2.

29 Права и ключи

Неделя 4

Цель: Решать, кому что можно, в одном месте и заранее.

Безопасность — это не «спрятать кнопку», а проверить право на входе в действие. Каждая программа получает **ключ** со списком того, что ей разрешено. Функция камеры сама проверяет ключ и отказывает без него.

В Omnium нет «администратора, которому можно всё». Программа может только то, на что ей дали ключ-канал. Нет канала к диску — нет доступа к диску вообще.

ПРИЛОЖЕНИЕ
Роли в CRM: работник видит свои карточки, админ — все, деньги — только суперадмин.

ИГРА
Сервер проверяет каждый ход игрока, а не верит тому, что прислал клиент.

ОС
Ключи Omnium: служба отвечает только тому, у кого есть её канал.

Паттерн «Проверка на входе». Право проверяет тот, кто выполняет действие, а не тот, кто показывает кнопку.

КОД: DAY29.TB

```
// день 29: права — ключ проверяют на входе
obj Key
    owner: text
    can: list text

func allowed(k: Key, action: text) gives ok: bool
    ok = false
    each c in k.can
        if c == action
            ok = true

func open_camera(k: Key)
    if not allowed(k, "камера")
        fail k.owner + ": нет ключа на камеру"
    print(k.owner, "снимает")

func main()
    chat = Key(owner: "чат", can: ["сеть", "камера"])
    game = Key(owner: "игра", can: ["звук"])
    try open_camera(chat)
    open_camera(game) else err
        print("отказано:", err)
```

ВЫВОД

```
чат снимает
отказано: игра: нет ключа на камеру
```

Задание. Добавьте ключу срок жизни `until: int` и отказывайте, если время вышло.

30 Общий скелет

Цель: Собрать всё в одну схему, по которой можно разобрать любую программу.

Посмотрите назад: у калькулятора, футбола и операционной системы один скелет. Есть **состояние** (модель, часто с режимом-автоматом). Есть **события**, которые его меняют. Есть **цикл**, который события ждёт или крутится по кадрам. И есть **слои**: вывод только читает, логика только считает, хранение только хранит.

Когда вы открываете чужой код, ищите эти четыре вещи. Когда пишете свой — начинайте с них: какие режимы, какие события, какой цикл, какие слои. Дальше остаётся заполнить функции.

ПРИЛОЖЕНИЕ
Список дел: модель задач, клики, цикл окна, слой сохранения.

ИГРА
Футбол: мир и счёт, ввод и голы, игровой цикл, физика отдельно от рисования.

ОС
Omnium: клетки и службы, прерывания и сообщения, цикл ядра, драйверы → ФС → оболочка.

Паттерн «Состояние + события + цикл + слои». Любую программу можно разложить на эти четыре части.

КОД: DAY30.TB

```
// день 30: общий скелет — состояние, события, цикл, слои
enum Mode
    work
    done

obj App
    mode: Mode
    todo: list text

func handle(a: App, ev: text)            // события меняют состояние
    if ev == "готово" and len(a.todo) > 0
        last = a.todo.pop()
    if len(a.todo) == 0
        a.mode = Mode.done

func draw(a: App)                        // вывод только читает
    print("режим:", a.mode, " дел:", len(a.todo))

func main()
    a = App(mode: Mode.work, todo: ["код", "тест", "отправить"])
    each ev in ["готово", "готово", "готово"]   // цикл событий
        handle(a, ev)
        draw(a)
```

ВЫВОД

```
режим: work  дел: 2
режим: work  дел: 1
режим: done  дел: 0
```

Задание. Финальный проект: выберите приложение, игру или маленькую службу и напишите её, начав с четырёх ответов на бумаге.

Шпаргалка: паттерны 15–30

День	Паттерн	Правило
15	**Игровой цикл**	Каждый кадр: dt → ввод → обновить → нарисовать. Скорость всегда × dt.
16	**Интегратор**	скорость += ускорение × dt; позиция += скорость × dt.
17	**Пересечение коробок**	Не пересекаются, если одна целиком с одной стороны. Иначе — касание.
18	**Список сущностей**	Обновить всех → собрать живых в новый список → заменить.
19	**Автомат экранов**	Режим программы — один enum. Каждый экран обрабатывает только свои события.
20	**Плавный переход**	Значение = lerp(было, стало, ease(t)). t идёт от 0 до 1 по времени.
21	**Очередь событий**	Источник только сообщает. Слушатели сами решают, что делать.
22	**Смотри → реши → действуй**	Правила от срочного к обычному. Первое подходящее и выигрывает.
23	**Служба**	Свои данные у каждого. Просьба — сообщение в канал. Ответ — сообщение обратно.
24	**Владение**	У каждого куска один хозяин. Взял — отдай ровно один раз.
25	**Раздели и собери**	Каждый работник — своя часть. Общее — shared и только под lock.
26	**Производитель / потребитель**	Между ними канал. Закрыл канал — значит работы больше нет.
27	**Кольцо**	Фиксированный массив, два указателя, всё по модулю длины.
28	**По кругу**	Очередь задач. Каждой — квант. Не закончила — в конец очереди.
29	**Проверка на входе**	Право проверяет тот, кто выполняет действие, а не тот, кто показывает кнопку.
30	**Состояние + события + цикл + слои**	Любую программу можно разложить на эти четыре части.

Что дальше

Теперь читайте настоящий код и ищите в нём знакомые паттерны.

Файл	Что там искать
`examples/apps/files`	Модель и вид, слои, цикл событий окна, справочники.
`examples/apps/football/main.tb`	Игровой цикл, физика, столкновения, автомат вратаря, простой ИИ бота.
`examples/apps/playground/main.tb`	Сущности, плавность из `lib/anim.tb`, камера, которая догоняет героя.
`lib/ui.tb`	Как устроена библиотека кнопок: состояние фокуса, события мыши, раскладка.
`omnium/clock.tb`, `keyboard.tb`	Службы с каналами: как часы и клавиатура отвечают другим клеткам.
`omnium/ofs.tb`	Загрузить, изменить, сохранить; журнал изменений на диске.
`omnium/shell.tb`, `desktop.tb`	Оболочка «Капля»: окна, пересечения, плавность, события.

> Главная мысль курса: перед кодом ответьте на четыре вопроса. Что программа помнит? Какие события её меняют? Какой у неё цикл? На какие слои она делится? Если ответы есть, код пишется почти сам.