# Дьяволы

Визуализация

Изменить частоту кадров
def main() -> None:
```
    # ...

    # ==================== SNAPSHOT TIMING
====================
    # Using set for O(1) lookup instead of list
    draw_times: set[int] = {t for t in range(0, TIMEPOINTS,
10)}  # ← ЭТО СТРОКА
    # ==================== END SNAPSHOT TIMING
====================
```

Инфекции

## РАЗНАЯ ЛОГИКА ФАЗ ИНФЕКЦИИ

**Версия 1 (TimeRunAndDeath()):**

```
python
# Фаза 1 → Фаза 2
mask = (AgeOfDisease <= incubation) * InfectionStatus.eq(1)
InfectionStatus[mask] = 2

# Фаза 2 → Фаза 1
mask = (AgeOfDisease > incubation) * InfectionStatus.eq(2)
InfectionStatus[mask] = 1
```

**Версия 2 (main.py):**

```
python
# Только Фаза 1 → Фаза 2 (однонаправленно!)
phase2_transition_mask = infected_phase1_mask &

(simulation_state.age_of_disease[:n] >=
DISEASE_PROGRESSION_THRESHOLD)
simulation_state.infection_status[:n] = th.where(
    phase2_transition_mask,
    INFECTION_STATUS_SYMPTOMATIC,  # = 2
    simulation_state.infection_status[:n]
)
# НЕТ обратного перехода Фаза 2 → Фаза 1!
```

Критическая разница:

- Версия 1: ЦИКЛИЧЕСКИЕ фазы (1→2→1→2...)
- Версия 2: ОДНОНАПРАВЛЕННЫЙ переход (1→2) и всё

## РАЗНАЯ ЛОГИКА ПОЛОВОЙ ПЕРЕДАЧИ

**Версия 1:**

```
python
if replication and (CurrentTime % 120 <= 10):
    # ДНИ 0–10: Только те, кто в фазе 1 (латентной) передают
половым путем
    phase1_transmitters =
(InfectionStatus[transmitter_indices] == 1)
```

**Версия 2:**

```
python
if day_in_year < breeding_days:  # breeding_days = 11
    # Только взрослые в фазе 1 передают половым путем
    phase1_transmitters =
(state×infection_status[transmitters] == 1)
```

**Разница: В версии 1 дополнительная проверка** replication **(флаг размножения).**

## РАЗНАЯ ОБРАБОТКА ВОЗРАСТА БОЛЕЗНИ

**Версия 1:**

```
python
AgeOfDisease = InfectionStatus.gt(0) + AgeOfDisease
# Увеличивает ТОЛЬКО для инфицированных (InfectionStatus >
0)
```

**Версия 2:**

```
python
simulation_state.age_of_disease[:n] +=
infected_phase1_mask.to(...)
# Увеличивает ТОЛЬКО для фазы 1 (infection_status == 1)
```

**Разница: В версии 1 возраст болезни растет у ВСЕХ инфицированных, в версии 2 - только у фазы 1.**

Движение

## РАЗНАЯ СИЛА ПРИВЯЗКИ К ТЕРРИТОРИИ

**Версия 1:**

```
python
# Сила привязки: тем сильнее, чем дальше от центра
territory_attraction = 0.05  # Фиксированная сила
```

```
# Применяем силу привязки (ОТТАЛКИВАНИЕ от центра)
speedX[has_territory] -= dist_to_center_x *
territory_attraction
#                  минус ^ = движение К центру
```

**Версия 2:**

```
python
# Расстояние от центра территории
dist_to_center_x = state.territory_center_x[:n] -
state.x[:n]  # ← Направление К центру
dist_to_center_y = state.territory_center_y[:n] -
state.y[:n]

# Attraction force proportional to distance from center
attraction_strength = 0.1  # Сила В 2 раза больше!

# Применяем силу (ПРИТЯЖЕНИЕ к центру)
state.speed_x[:n][resident_mask] +=
dist_to_center_x[resident_mask] * attraction_strength
#                  плюс ^ = движение К центру
```

**Разница:**

- Версия 1: -= (отрицательная сила) = движение к центру
- Версия 2: += (положительная сила) = движение к центру
- Сила: 0.05 vs 0.1 (в 2 раза сильнее в версии 2)

**Физически одинаково, но формулы противоположны по знаку.**

## РАЗНАЯ ЛОГИКА ОГРАНИЧЕНИЯ ТЕРРИТОРИИ

**Версия 1:**

```
python
# Если вышли за пределы территории
outside_territory = dist_to_center > Range
if outside_territory.any():
    # 1. Возвращаем к границе территории
    correction_factor = Range /
dist_to_center[outside_territory]
    X_correction = territory_center_x[has_territory]
[outside_territory] + \
                   dist_to_center_x[outside_territory] *
correction_factor

    # 2. Обнуляем скорость в направлении от центра
    for idx in correction_indices:
        dx = X[idx] - territory_center_x[idx]
```

```
        dy = Y[idx] - territory_center_y[idx]
        if dx != 0 or dy != 0:
            dir_magnitude = th.sqrt(dx**2 + dy**2)
            dir_x = dx / dir_magnitude
            dir_y = dy / dir_magnitude

            # Проекция и уменьшение скорости
            speed_proj = speedX[idx] * dir_x + speedY[idx] *
dir_y
            if speed_proj > 0:  # Если скорость направлена
от центра
                speedX[idx] -= dir_x * speed_proj * 0.5
                speedY[idx] -= dir_y * speed_proj * 0.5
```

**Версия 2:**

```
python
# Если вышли за пределы территории
outside_territory = (dist_from_center > RANGE) &
resident_mask
if outside_territory.any():
    # Просто телепортируем к границе территории
    dir_x = state.x[:n] - state.territory_center_x[:n]
    dir_y = state.y[:n] - state.territory_center_y[:n]

    state.x[:n][outside_territory] = (
        state.territory_center_x[:n][outside_territory] +
        dir_x[outside_territory] /
dist_safe[outside_territory] * RANGE
    )
    # НЕТ коррекции скорости!
```

**Критическая разница:**

- Версия 1: Возвращает к границе + уменьшает скорость от центра
- Версия 2: Просто телепортирует к границе, скорость не меняется

Генетика

## РАЗНАЯ ЛОГИКА НАСЛЕДОВАНИЯ ГЕНОВ

**Версия 1 (chrom_cal()):**

```
python
# СЛОЖНАЯ логика выбора аллелей:
for f_idx in range(female_number):
    mother_geno = f_a[f_idx]  # [аллель1, аллель2]
```

```
    allele1, allele2 = mother_geno[0].item(),
mother_geno[1].item()

    for child_idx in range(numofprogeny):
        if allele1 == 0 and allele2 == 0:  # [0,0] → всегда
0
            female_a[f_idx, child_idx] = 0
        elif allele1 == 1 and allele2 == 1:  # [1,1] →
всегда 1
            female_a[f_idx, child_idx] = 1
        else:  # [0,1] или [1,0] → случайный аллель
            female_a[f_idx, child_idx] = th.randint(0, 2,
(1,), device=device)×item()
```

**Версия 2 (calculate_chrom()):**

```
python
# ПРОСТАЯ детерминированная логика:
# Всегда берет ПЕРВЫЙ аллель от каждого родителя!
mother_alleles = female_repro_allele[:,
0].unsqueeze(1).expand(-1, num_progeny)
father_alleles = male_repro_allele[:,
0].unsqueeze(1).expand(-1, num_progeny)
repro_alleles = th.stack([mother_alleles, father_alleles],
dim=2)
```

**Критическая разница:**

- Версия 1: Менделевское наследование (случайный выбор аллеля от гетерозигот)
- Версия 2: Всегда первый аллель (детерминированное, не биологичное)

## РАЗНАЯ ЛОГИКА ВЫБОРА ПОЛА

**Версия 1:**

```
python
# Пол определяется X/Y аллелями:
# Мать: ВСЕГДА X-аллель (0)
mother_allele_for_sex[f_idx, :] = 0  # X от матери

# Отец: случайно X (0) или Y (1)
for child_idx in range(numofprogeny):
    father_allele_for_sex[f_idx, child_idx] = th.randint(0,
2, (1,), device=device).item()

# Потомок: XX = самка (0+0), XY = самец (0+1)
children_sex = all_children_sex_alleles.sum(1)
```

**Версия 2:**

```
python
# Пол определяется ВТОРЫМ аллелем от отца:
# Мать: всегда первый аллель (0 = X)
mother_sex = female_sex_chrom[:, 0].unsqueeze(1).expand(-1,
num_progeny)

# Отец: всегда второй аллель (0 = X, 1 = Y)
father_sex = male_sex_chrom[:, 1].unsqueeze(1).expand(-1,
num_progeny)

# Если второй аллель отца = 1 → самец, иначе → самка
offspring_sex = offspring_chrom_sex[:, 1] == 1  # True =
male, False = female
```

**Разница:**

- Версия 1: Случайный выбор X/Y от отца
- Версия 2: Детерминированный - всегда берет второй аллель отца

ДА, в версии 2 ВСЕГДА рождаются самцы!

Смертность:

**РАЗНАЯ ЛОГИКА СМЕРТИ ЮВЕНИЛОВ БЕЗ ТЕРРИТОРИИ**

## Версия 1 (TimeRunAndDeath()):

```
python
# Смерть ювенилов без территории:
juv_no_terr_mask = (status == STATUS_JUVENILE_NO_TERR)
if juv_no_terr_mask.any():
    juv_fitness = Fitness[juv_no_terr_mask]
    fitness_factor = th.nn.functional.relu(80 -
juv_fitness) / 80

    age_over_deadline = (Age[juv_no_terr_mask] >
dispersalDeadline)  # = 160
    death_prob = fitness_factor * age_over_deadline.float()
```

## Версия 2 (process_all_deaths()):

```
python
# Probabilistic death between dispersal_deadline and
maturity_age
prob_death_candidates: Tensor = juv_no_terr_mask &
```

```
                                (ages > dispersal_deadline)
&  # DISPERSAL_DEADLINE = 160
                                (ages < maturity_age)
# AGE_JUVENILE_TO_ADULT = 220

# fitness_factor: lower fitness = higher death chance (0 to
1)
fitness_factor: Tensor = th.clamp(1.0 - (fitness / 100.0),
min=0.0, max=1.0)
```

**Критическая разница в формуле фитнеса:**

- Версия 1: (80 - fitness) / 80 → чем МЕНЬШЕ фитнес, тем ВЫШЕ смертность
- Версия 2: 1.0 - (fitness / 100.0) → тоже чем меньше фитнес, тем выше смертность

**Но численно РАЗНЫЕ формулы! Для fitness=60:**

- Версия 1: (80-60)/80 = 0.25
- Версия 2: 1.0 - (60/100) = 0.40

**Версия 2 дает ВДВОЕ БОЛЬШУЮ смертность при том же фитнесе!**

Расселение:

**РАЗНАЯ ОБРАБОТКА DISPERSAL (расселение)**

## Версия 1 (DisperseJuvenile()):

```
python
# Вызывается в день 100 (timeOfDisposal)
# Рождает потомство от матрицы размножения
# Добавляет случайное смещение ±50
```

## Версия 2 (disperse_offspring()):

```
python
def disperse_offspring(state, day_in_year,
time_of_disposal=TIME_OF_DISPOSAL):
    if day_in_year == time_of_disposal:  # день 100
        # Находит ВСЕХ детей и меняет их статус
        to_disperse = th×nonzero((state×status[:n] ==
STATUS_CHILD))[0]
        state.status[to_disperse] = STATUS_JUVENILE_NO_TERR
        state.age[to_disperse] = AGE_CHILD_AFTER_DISPOSAL  #
= 85
```

**Критическая разница:**

- Версия 1: DisperseJuvenile = рождение НОВОГО потомства

- Версия 2: disperse_offspring = изменение статуса УЖЕ СУЩЕСТВУЮЩИХ детей

**Это РАЗНЫЕ ПОНЯТИЯ! В версии 2 дети уже существуют (родились ранее), а в версии 1 они рождаются в день 100.**

ВРЕМЕННАЯ ЛИНИЯ ВЕРСИИ 1:

| Дни 0–10 | Дни 11–99 | День 100 | После 100 |
|---|---|---|---|
| Размножение | Беременность | ↓ РОЖДЕНИЕ! Дети рожда- ются с воз- растом 85 | Детство Дети растут (85+ дней) |

ВРЕМЕННАЯ ЛИНИЯ ВЕРСИИ 2:

| Дни 0–10 | Дни 11–99 | День 100 | После 100 |
|---|---|---|---|
| Размножение ↓ РОЖДЕНИЕ! Дети рожда- ются с воз- растом 0 | ↓ Дети уже существуют! Но в "логове" (невидимы) | ↓ РАССЕЛЕНИЕ Меняют статус на "ювенилы" и получают возраст 85 | Детство Дети растут (85+ дней) |

1. Координаты рождения:
   - Версия 1: Мать + случайное смещение ±50
   - Версия 2: Точно как у матери (без смещения!)
2. Время "детства":
   - Версия 1: Дети сразу 85 дней → статус "ребенок" (0-159 дней)
   - Версия 2: Дети 0 дней → мгновенно 85 дней → статус "ювенил без территории"
3. Фактический статус после дня 100:
   - Версия 1: STATUS_CHILD (потому что < 160 дней)

- Версия 2: STATUS_JUVENILE_NO_TERR (потому что disperse_offspring меняет статус)

**ВЛИЯНИЕ НА СИМУЛЯЦИЮ:**

1. Пространственное распределение:
   - Версия 1: Дети разбросаны вокруг матери (±50)
   - Версия 2: Все дети в одной точке (координаты матери)
2. Возрастная структура:
   - Версия 1: Есть "дети" (85-159 дней)
   - Версия 2: НЕТ "детей"! Сразу "ювенилы без территории"
3. Движение и инфекция:
   - Версия 1: Дети двигаются (статус = ребенок)
   - Версия 2: "Дети" сразу ювенилы, могут искать территорию

Это ОГРОМНАЯ разница! В версии 2 пропущен этап детства, дети сразу становятся ювенилами!