Мемы, шейдеры, оверинжиниринг и PixiJS: как полноценная RTS-игра в вебе грузится за 3 секунды и выдает 120 FPS

NOV 2024

Что такое Age of Memes?

https://youtu.be/keRMAUpgTOc

https://youtu.be/aBZGAq9fm8c

Интро

Я - Данила.

- В разработке 15 лет, начинал с ботов и альтернативного клиента для Lineage 2
- Сделал в 2019 клон Figma на WebGL + Skia + JS/C++ (WASM)
- Писал игры на Unity

Почему вообще веб?

- Мгновенная дистрибуция - юзер ткнул ссылку и начал играть
- Мгновенный мультиплеер - кинул ссылку другу и начали играть
- Мгновенные апдейты игры - залил новые текстуры - у всех они есть

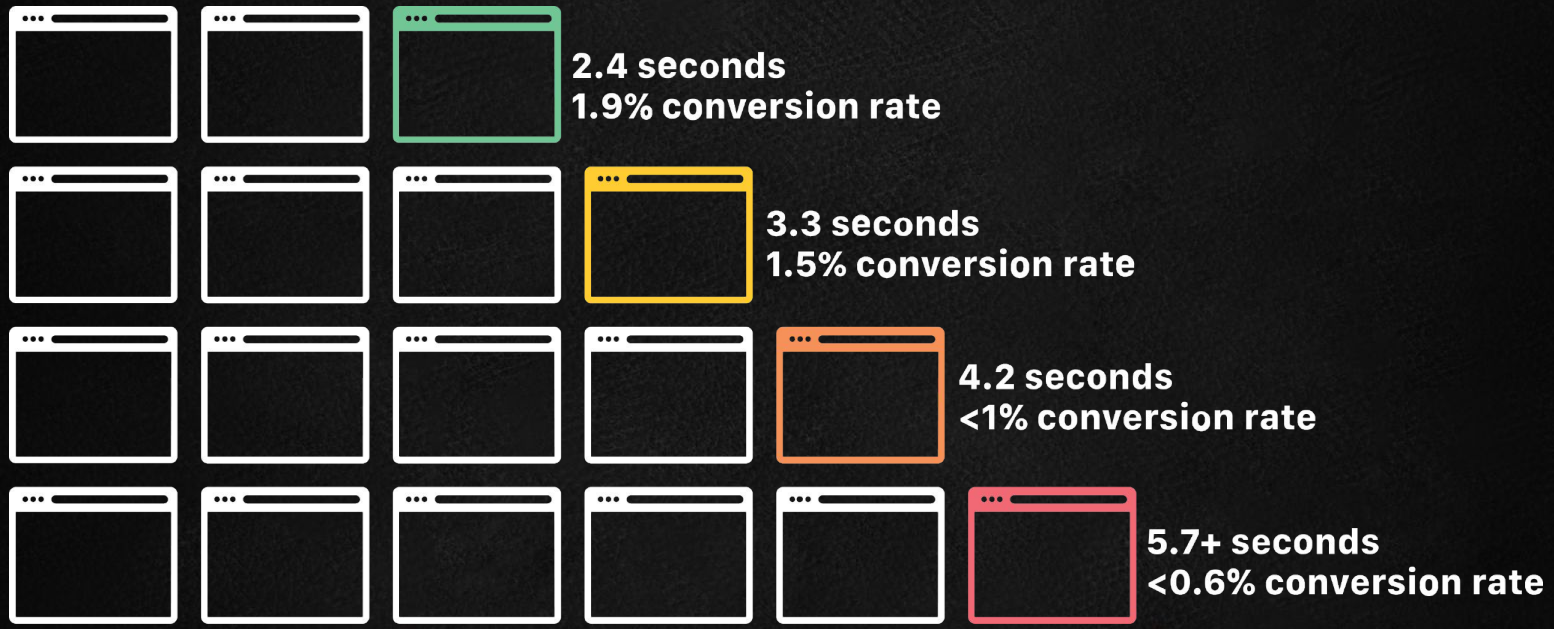
https://a16zgames.substack.com/p/why-web-gaming-died-and-how-its-coming-back

Почему вообще веб?

- Рендерниг и вычисления в вебе невероятно производительные
- Никаких ограничений связанных с магазинами игр
- Стереотип “веб для флеш игр” уходит в прошлое

Главное отличие веба

Если что-то не загрузилось за 5-10 секунд - вкладка закрывается.



А что по требованиям для такой игры?

RECOMMENDED:

OS: Windows 10 64bit

Processor: 2.4 Ghz i5 or greater or AMD equivalent

Memory: 8 GB RAM

Graphics: Nvidia® GTX 650 or AMD HD 5850 or better

DirectX: Version 11

Network: Broadband Internet connection

Storage: 15 GB available space

Э?

Storage: 15 GB available space

за 5-10 секунд - вкладка закрывается.


700
1000
1030
1000
1000
3

Итого

- Сжимаем всё так сильно, как только можем
- Грузим только то, что нужно прямо сейчас на экране
- Оптимизируем рендеринг по максимуму

О чем доклад?

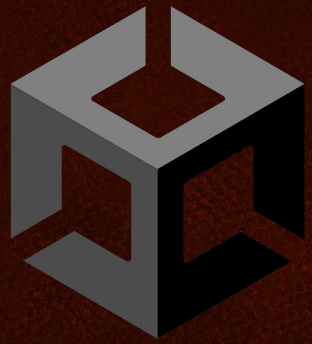
Как мы сделали свой игровой движок на базе PixiJS, и на нём сделали игру Age of Memes. А именно:

- Обзор существующих игровых движков в вебе и почему сделали свой
- Как мы сжимали и упаковывали всё, что только можно
- Как мы учились грузить только то, что нужно
- Как мы рендерим так, чтобы было быстро

Игровые движки для веба

Топ игровые движки для веб-игр

- Unity
- Godot
- Phaser.js
- Defold
- Cocos 2d

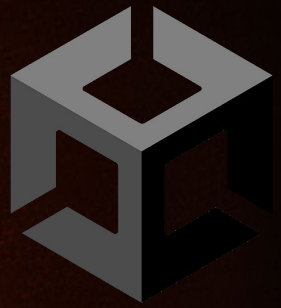


GODOT
Game engine

DEFOLD

COCOS2D

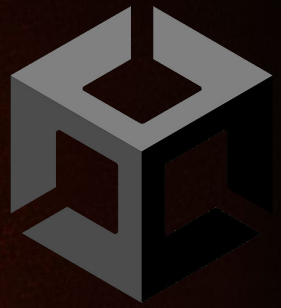
Unity - плюсы



- Лучшие возможности для рендеринга


5 / 15
2:33
Local Player
Settlement Level 1
2.8k / 2.8k
Dev
(0)
Settle ment Level 1
400
90
300
0
Hero Greek Jason
Hero Greek Odyss eus
Mason s
Olympi c Parent age
Age 3 Apollo
Age 3 Dionysos

Unity - минусы



- Худшая загрузка и размер бандла, наисложнейшее встраивание progressive loading пайплайнов (closed source)

Unity - минусы

- 7 мегабайт в сжатом виде: ПУСТОЙ 2d проект (Unity 6)

Danila Simonov edited 21:08

короче, последняя версия unity в максимальном режиме, делаю webgl билд
30.8 мегов получилось, в сжатом виде - 6.8 мегов.
это абсолютно пустой новосозданный 2д проект, без единой текстуры и всего такого, чисто код

Godot - плюсы

GODOT
Game engine

- Нормальный рендеринг, нормальный размер бандлов

Godot - минусы

- Наисложнейшее встраивание progressive loading пайплайнов
- Очень сложное редактирование движка (C++)

Phaser.js - плюсы

- Маленькие бандлы
- Matter-js - отличный движок физики
- Isometric Tilemaps есть

Phaser.js - минусы

- Оч простенькие 2д-игры, заточка под Playable Ads
- Хорошей работы с изометрией нет, рендеринг руками
- Phaser Editor - closed-source, с тех пор как поглощён Phaser
- Сложный progressive loading

Defold - плюсы

- Маленькие бандлы
- В общем-то и всё :(

Defold - минусы

- C++ движок
- Lua язык
- На нём нет нормальных игр

Cocos 2d - плюсы

- Маленькие бандлы
- И снова - всё :(

Cocos 2d - минусы

- C++ движок
- Lua язык
- Выкуплен китайцами, последний коммит - год назад

Итого, почему свой:

- Нужен мультиплеер с переиспользованием кода на фронте и сервере
- Нужна быстрая разработка, а не C++

Итого, почему свой:

- Нужна возможность глубоко редачить движок для оптимизаций
- Нужна физика оптимизированная именно под наши задачки (3д-стрелы, ядра)

Итого, почему свой:

- Нужен офигенный собственный редактор карт и уровней, потому что у нас бизнес-модель вокруг UGC

Жмём текстуры

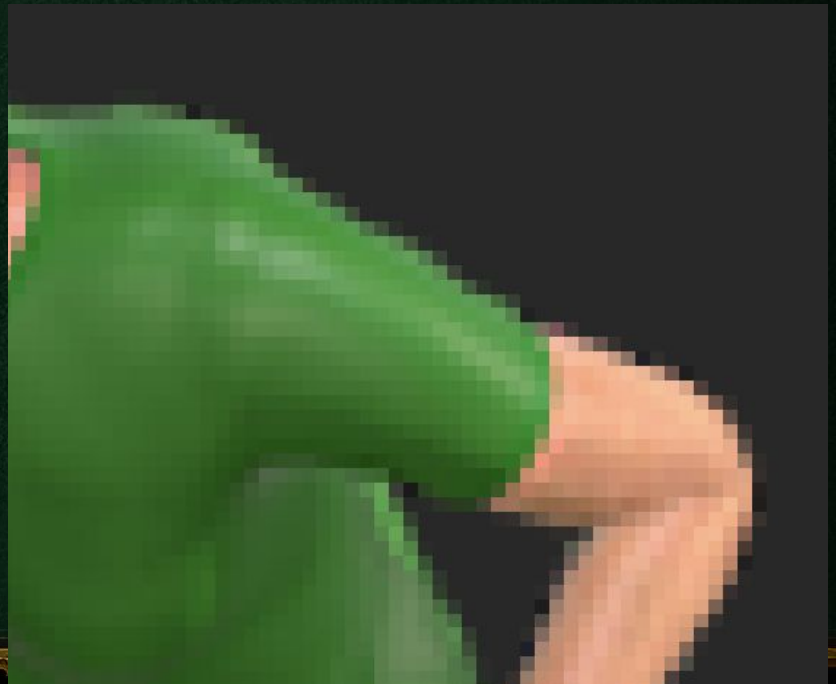
Как рендерить?

- Сразу маленький размер или ресайзить?
- Сразу маленький - быстрый рендер, низкое качество
- Ресайзить - долгий рендер, высокое качество

Рендерим с маленьким разрешением

В блендере

Рендер 256x256



Рендерим с большим разрешением

В блендере

На выходе 1024 -> 256

Рендерим послойно

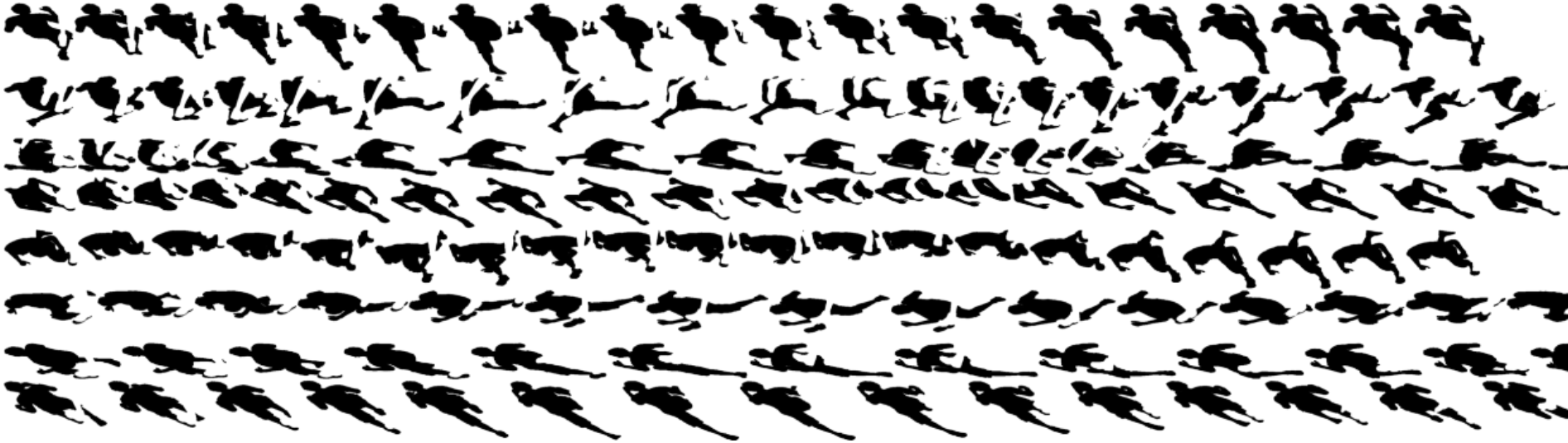
- Патчим блендер, чтобы экспортить слои по отдельности: тени, маски, etc

Рендерим послойно

32-bit webp

Рендерим послойно

8-bit PNG, x0.5 размер



Как ресайзить?

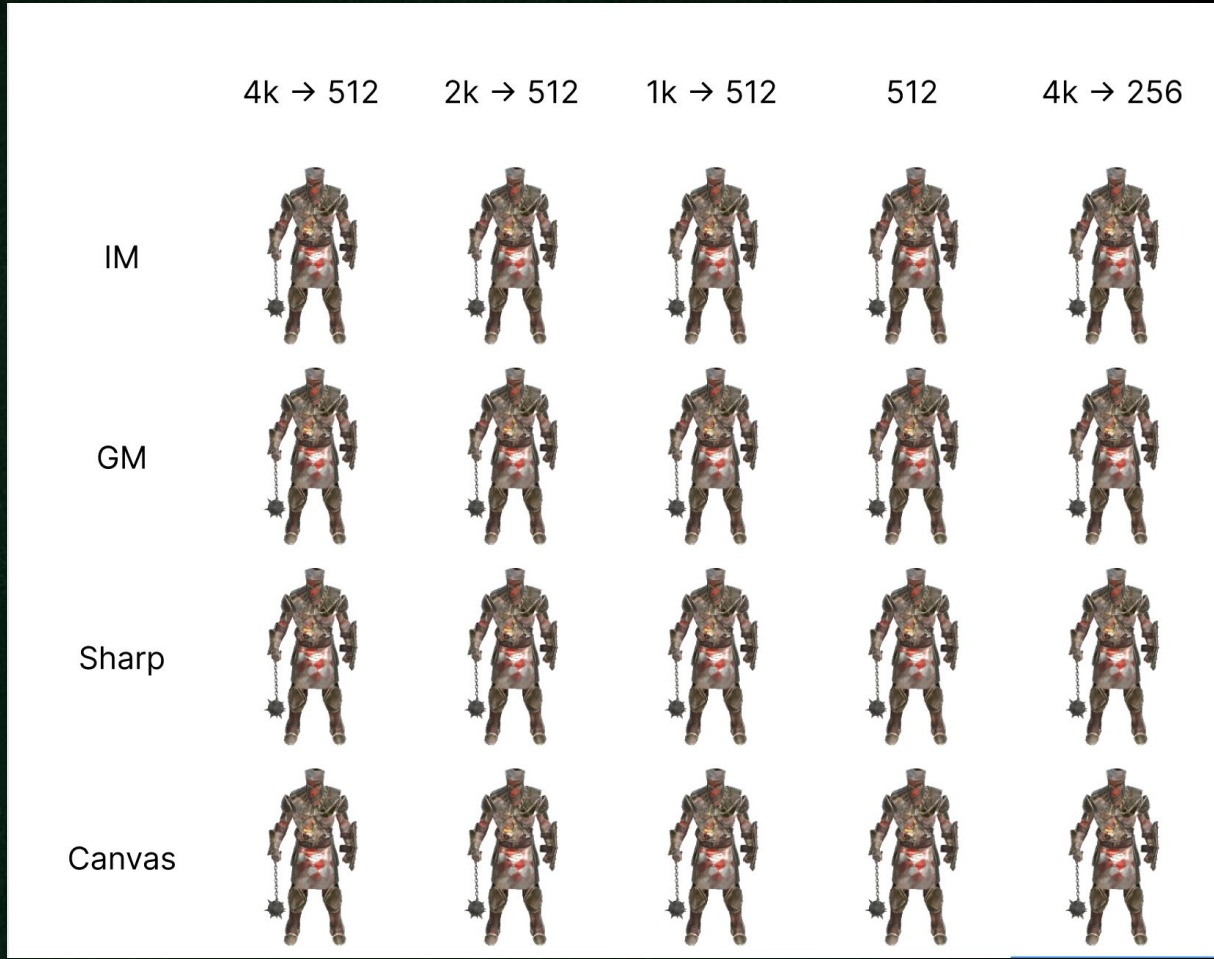
- ImageMagick
- GraphicsMagick
- Sharp
- Canvas
- Skia-Canvas
- Photoshop

Как ресайзить?

- 4k -> 256?
- 2k -> 256?
- 4k -> 512?
- 1k -> 256 / 512?
- А как быстрее? А как качественнее? А что по пост-фильтрам?
- А как пост-фильтры сохраняют консистентность между фреймами?

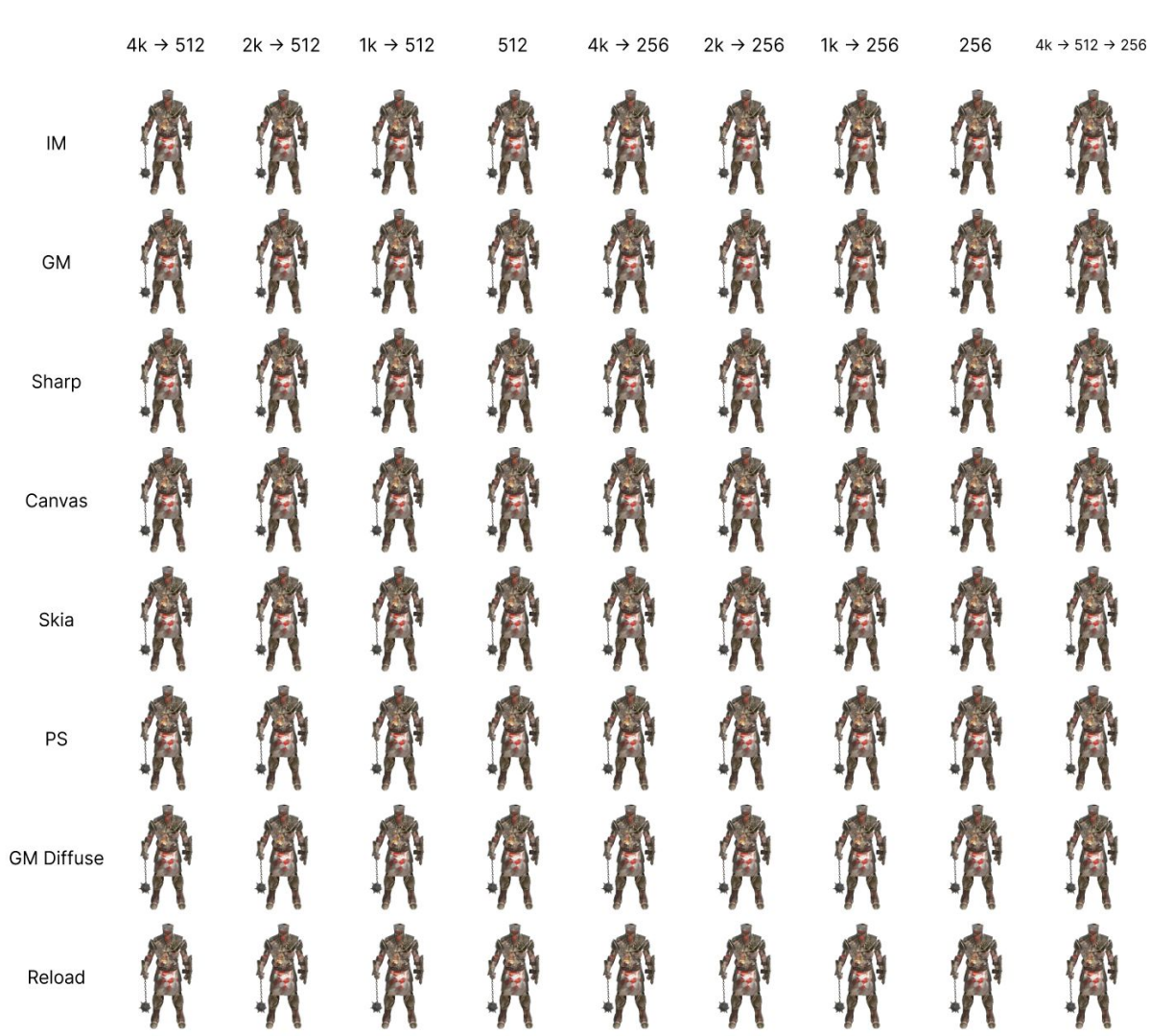
Как ресайзить?

- А точно ли есть универсальный one-size-fits-all способ?



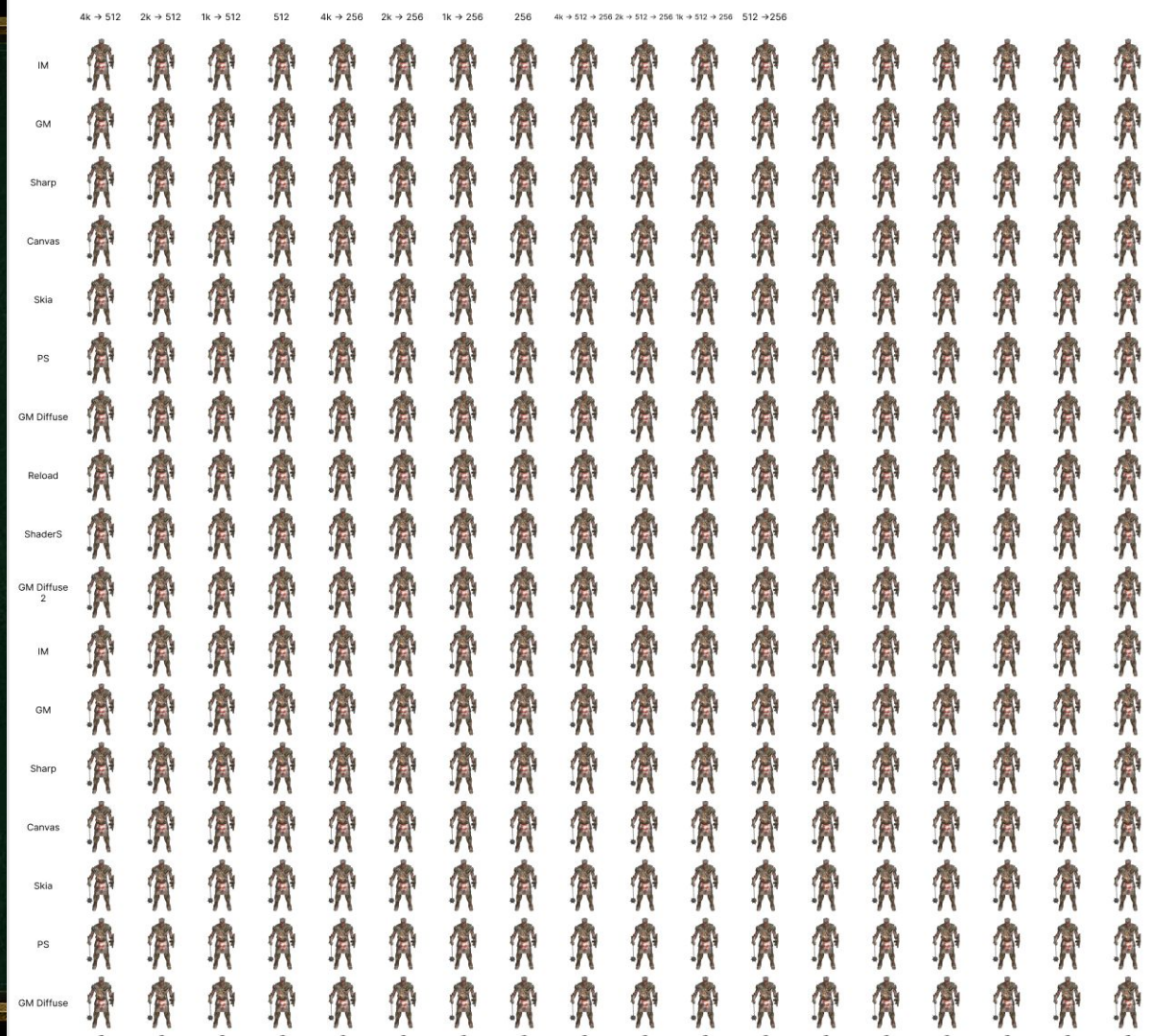
Как ресайзить?

- А точно ли есть универсальный one-size-fits-all способ?



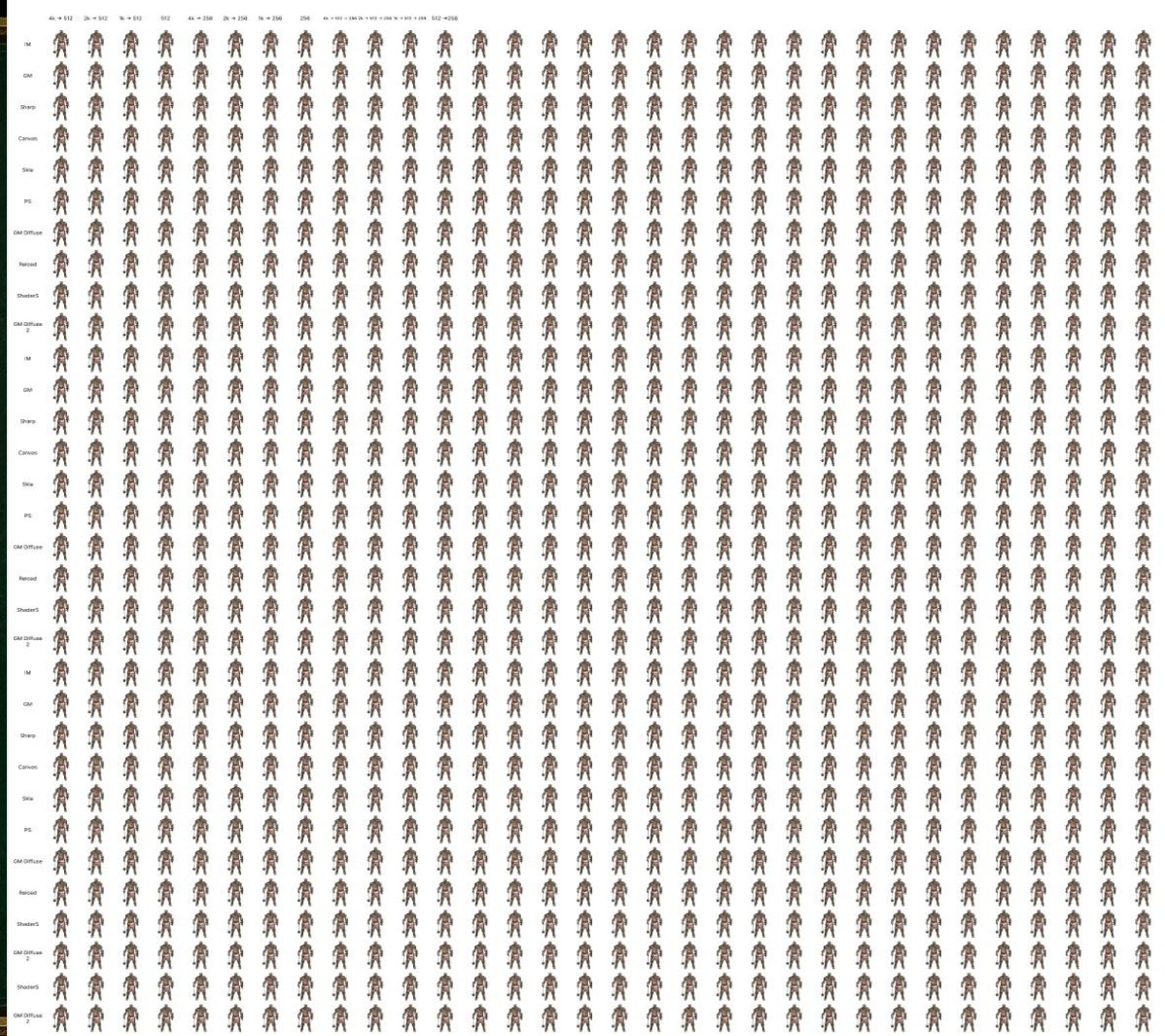
Как ресайзить?

- А точно ли есть универсальный one-size-fits-all способ?



Как ресайзить?

- А точно ли есть универсальный one-size-fits-all способ?



Как ресайзить: что выбрали

- ImageMagick - медленный, но лучшее качество и сохранение деталей

convert -sharpen 0x2.0 -distort Resize

Как сжимать?

- png? webp? jpg с шейдером? gif?
- 8-bit png для теней и масок + PNGOUT + OxiPNG + AdvPNG
- 32-bit png + ImageOptim + lossless/lossy webp для основных изображений

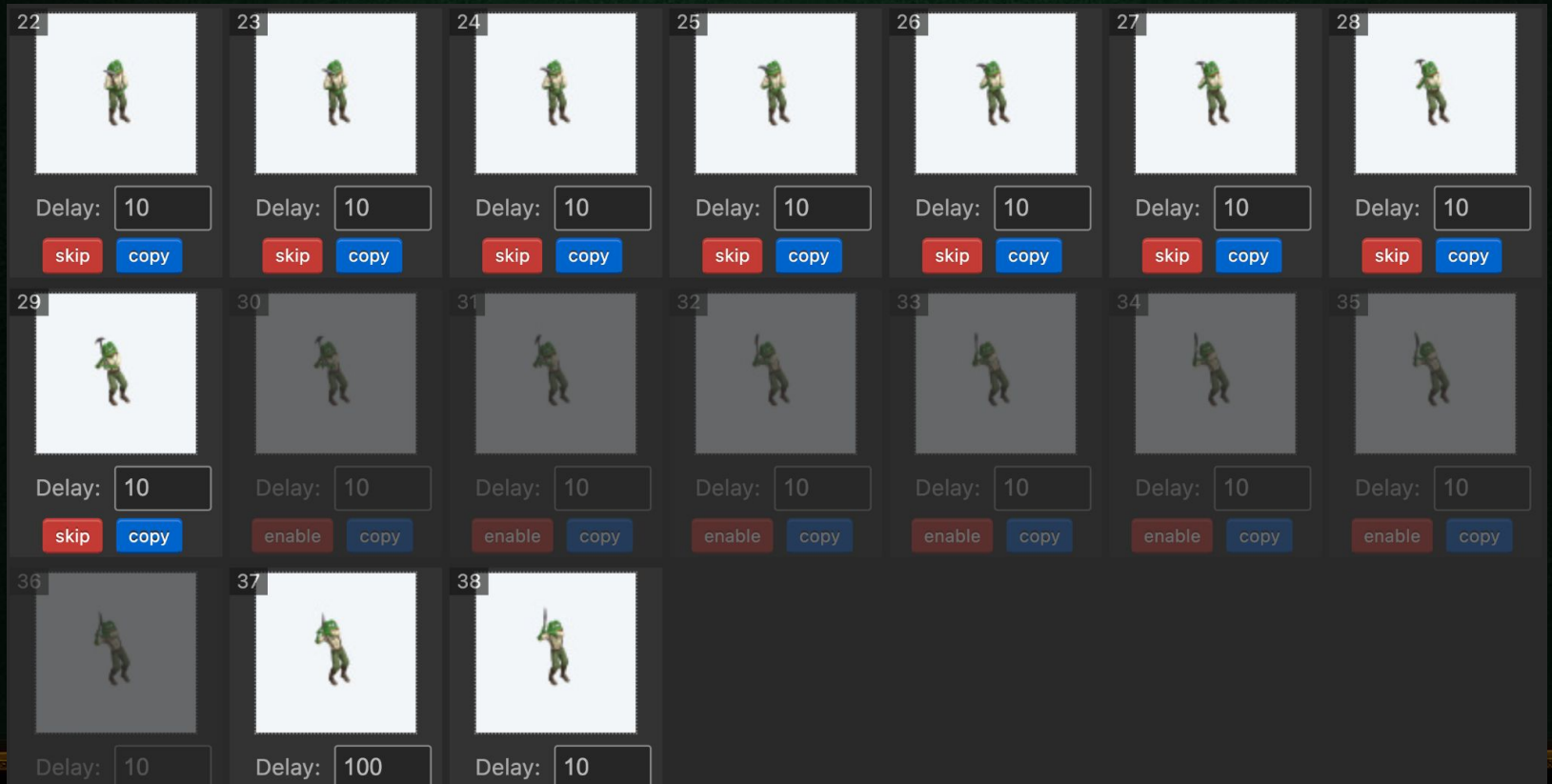
Как сжимать?

- Reduced-версии, меньше углов, меньше размер
- Перебор вариантов для каждого персонажа?

Управляем кадрами (динамический FPS)

- У анимациях на движениях кадров надо больше, а при статике - меньше

Мини-редактор чтобы выкидывать кадры



Компилируем разные версии

- iPhone Retina - x3 DPI для текстур, Retina - x2 DPI, Regular - x1 DPI

Компилируем разные версии

- Reduced - половина кадров/половина углов/половина разрешения

Reduced: До

Reduced: После

Делаем свою Ops-систему

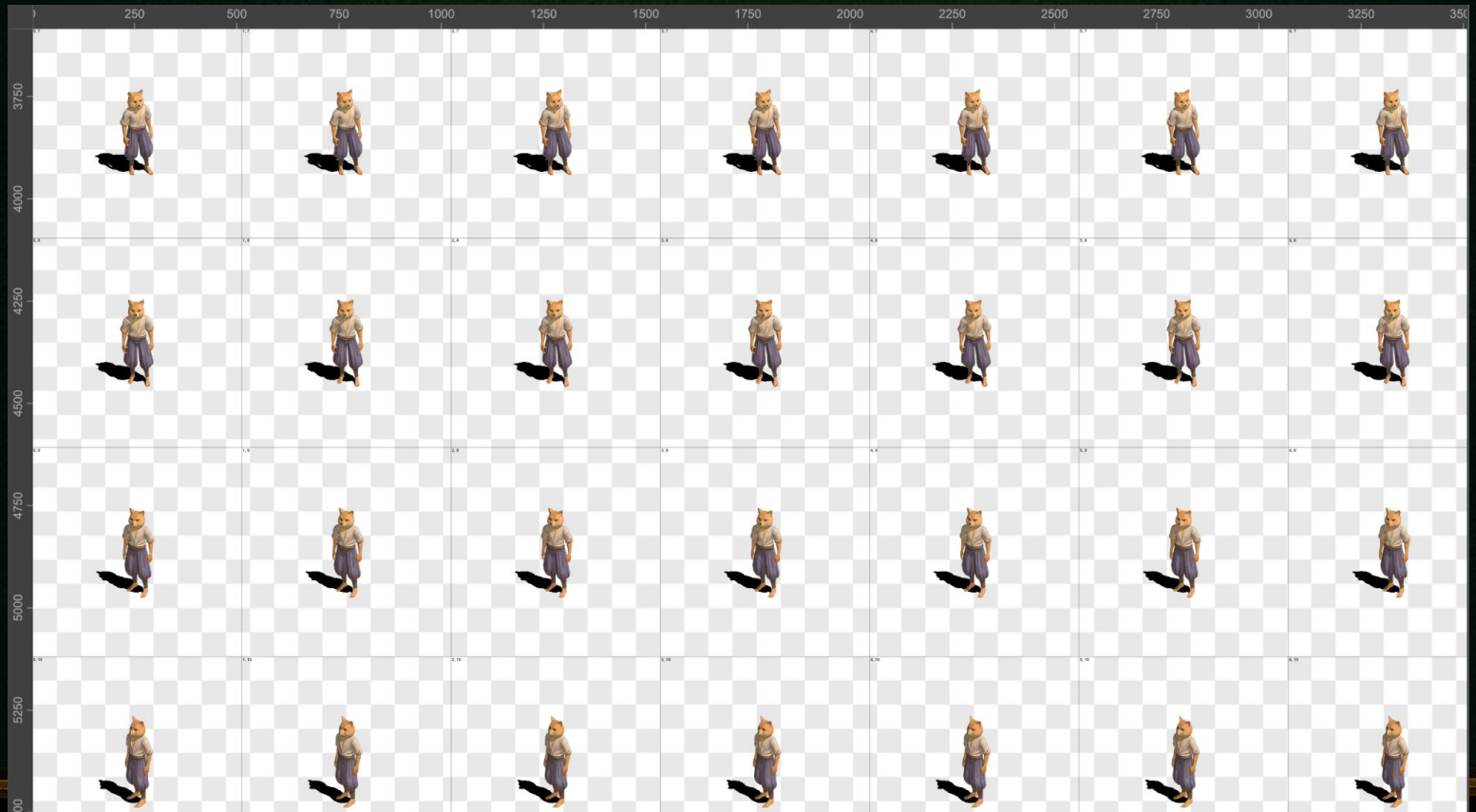
- Маленькая рендеринг-ферма - экспортируем сырые фреймы, после чего настраиваем все параметры ресайзов и сжатий

- Для новых персонажей делаем перебор вариантов для пары кадров, сохраняем настройки экспорта и сжатия

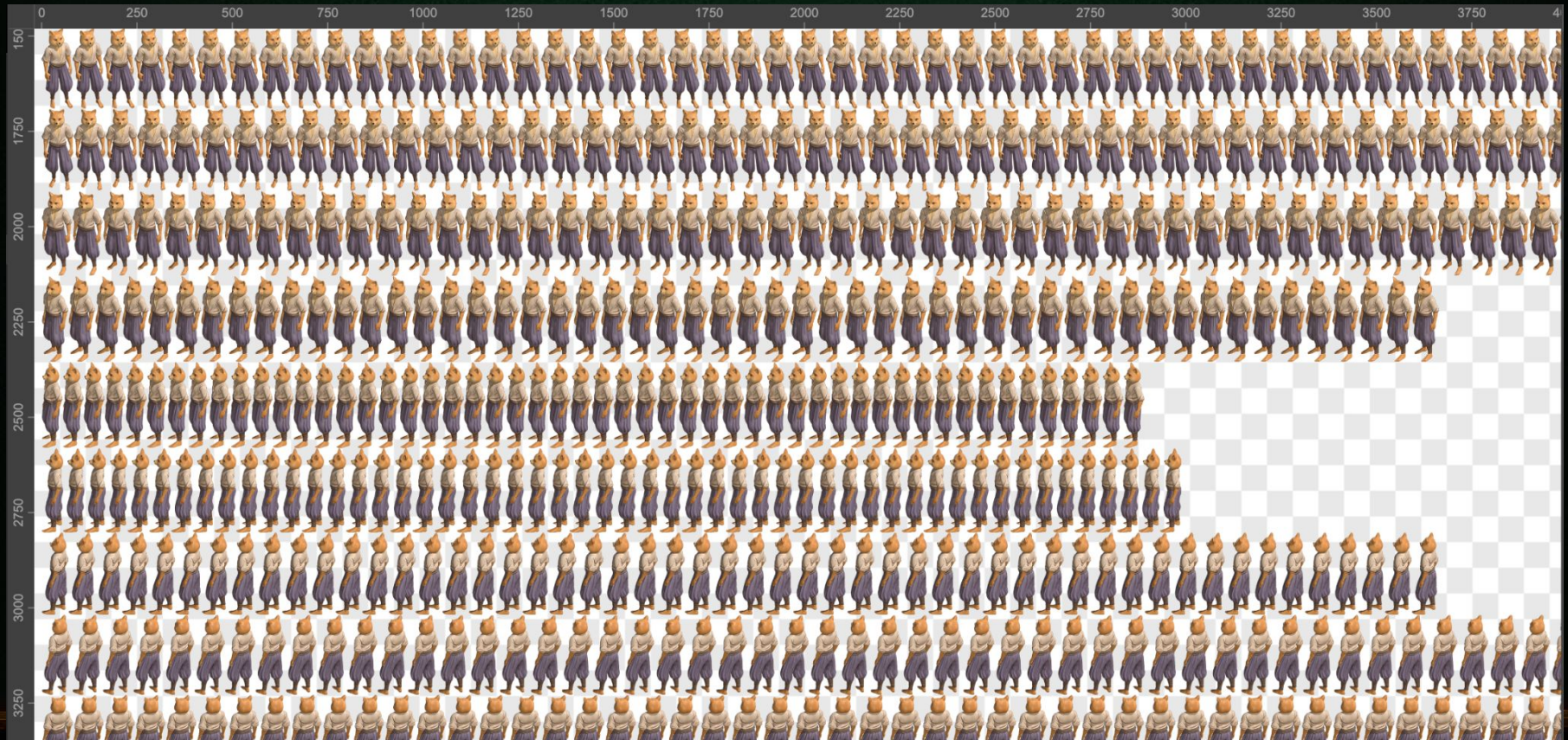
Делаем свою Ops-систему

- Экономим до 70-90% на одном спрайтшите
- Шерим кодовую базу между клиентом, сервером, редактором и аниматором - сразу экспортируем в формат редактора (.siv.zip)

Уплотняем Spritesheet: До



Уплотняем Spritesheet: После



Уплотняем Spritesheet

- spritesheet.js - пробовали другие варианты, пробовали сделать сами, в итоге пока что он лучший

Грузим только то, что нужно

Структура игровых ассетов

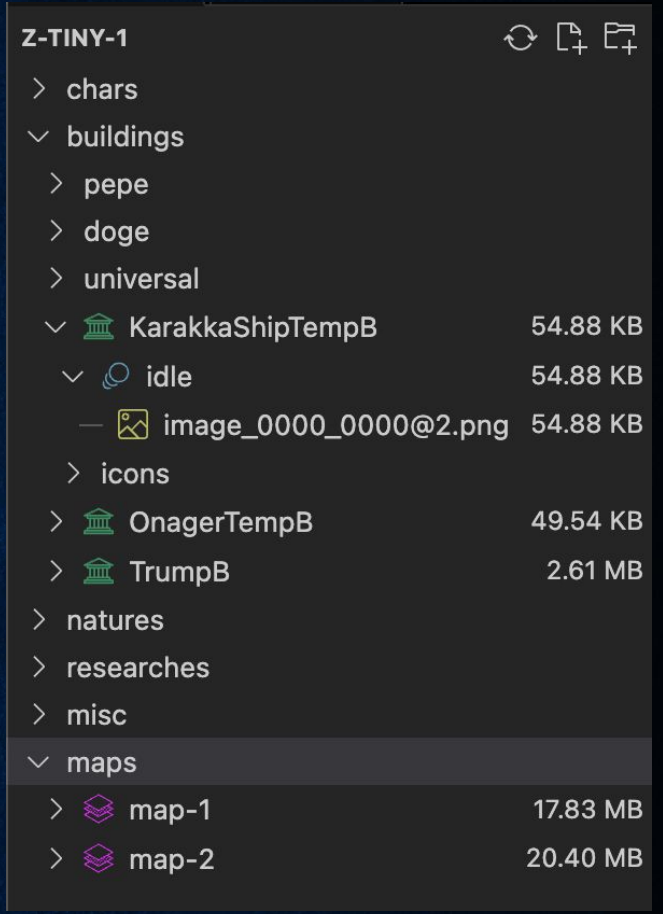
- Есть библиотеки - коллекции всех юнитов в рамках одной системы графики и игрового баланса (проще говоря - одной игры)

Пример библиотеки Age of Memes

Пример библиотеки Fantasy Kingdom

Структура игровых ассетов

- А есть карты - это “сцены” - конкретные уровни, которые проходит игрок
- И библиотеки, и карты - это простые сериализуемые объекты, с метаданными необходимыми для игрового процесса



Учим игру работать без текстур

- Метаданных должно быть достаточно для старта игры, даже если ни один ассет не загружен (headless-состояние)

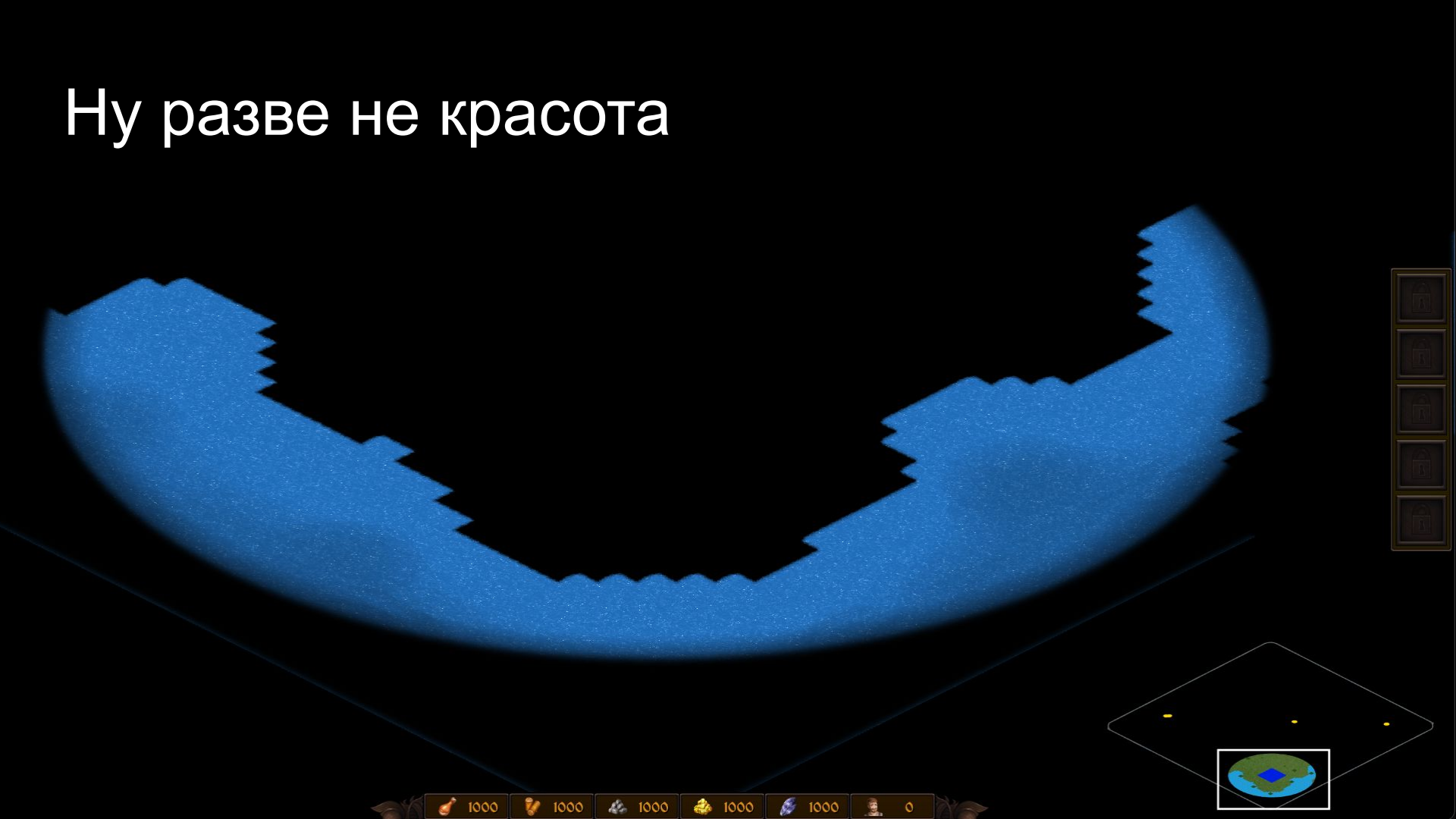
Учим игру работать без текстур

- Pixi.js прекрасно умеет в это из коробки
- Для собственных шейдеров приходилось допиливать подписки на статус загрузки текстур

Учим игру работать без текстур

- Теперь скорость запуска игры = скорость загрузки метаданных и JS-кода

Ну разве не красота



Приоритет загрузки

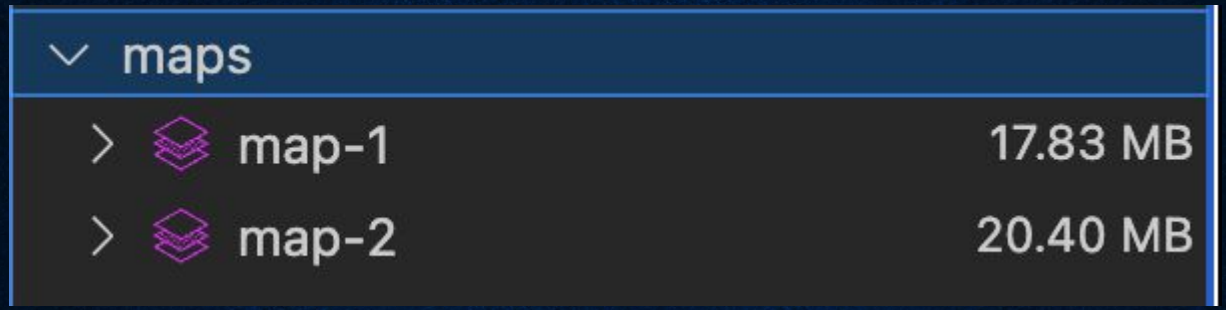
- На основе положения базы игрока, заранее считаем какие юниты будут у него на экране при старте игры (с запасом)
- Для каждого игрока - это первый приоритет, first contentful paint

Приоритет загрузки

- На основе данных о древе прокачки определяем, какие юниты будут доступны в первые 3-4 минуты игры - это второй приоритет
- Все остальные - третий приоритет

Приоритет загрузки

- Рядом с каждой картой в редакторе явно выводим размер в мегабайтах - сколько весят reduced-ассеты первого и второго приоритетов



Loader пока не загружены reduced версии для первого и второго приоритета

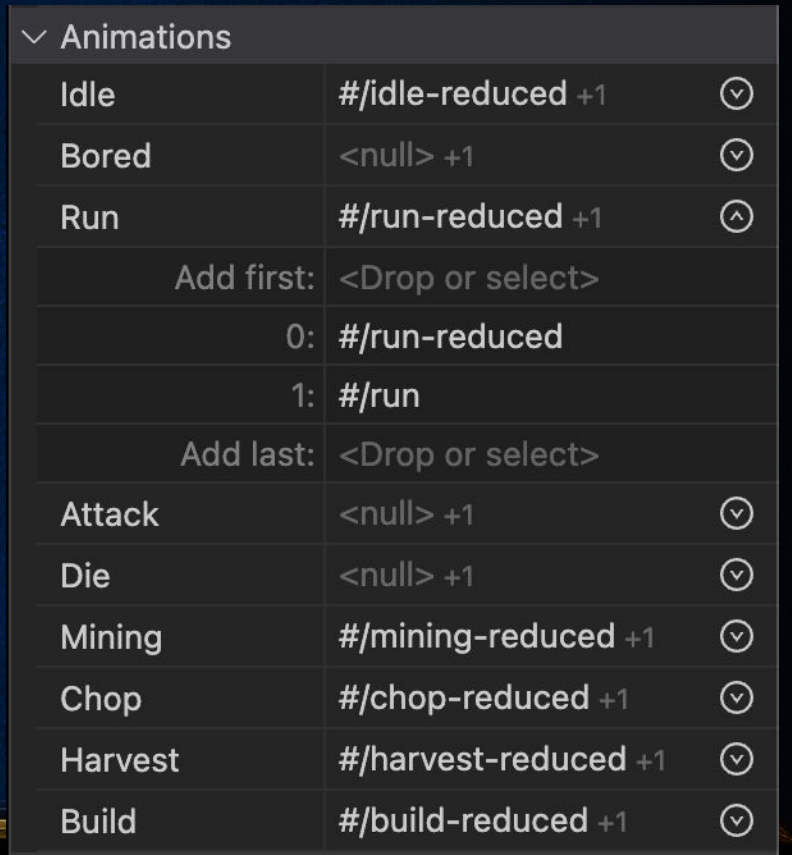
После этого - запуск, в фоне идёт идёт загрузка HD-версий первого и второго приоритета, они на лету подменяют reduced-версии

Как грузим приоритеты

- Только после полной прогрузки HD для 1 и 2 приоритетов, загружаются HD-версии остального контента внутри игры
- Контент одной карты не должен быть больше 300 мегабайт (1.6мб/с скорость интернета для загрузки 300МБ за 3 минуты)

Мануальные приоритеты в редакторе

- Иногда в карте есть логика нарушающая стандартный порядок – к примеру, late-stage юнит появится в начале игры.
- В редакторе на этот случай есть управление приоритетами загрузки с возможностью переназначать их



Динамический выбор DPI текстур

- Не надо грузить x3 DPI только потому, что устройство его поддерживает - важен фактический размер ассета на экране

Ньюансы Pixi.js (v7.x)

- Из коробки CREATE_IMAGE_BITMAP = false
- Без битмапов декодинг текстур идёт силами потока с gl-контекстом и блокирует рендеринг весьма заметно

Ньюансы Pixi.js (v7.x)

- С битмапами загрузка 4гб текстур на видеокарту происходит вообще незаметно для потока рендеринга, но…
- Битмапы в Pixi.js v7.x чутка забагованы и грузят картинку дважды (один раз image.src, второй - fetch+blob для createImageBitmap)

Ньюансы Pixi.js (v7.x)

- Не забываем про preserveBitmap = false, чтобы не держать битмапы в памяти

Рендерим так,
чтобы было быстро

Рендеринг: общие принципы

- Рендерим только то что на экране (свой B-tree-like spatial index)
- Режем на слои, чтобы не нарушать batching Pixi.js (он топ)

Рендеринг: общие принципы

- Режем на около-статичные и динамические слои (деревья меняются редко, персонажи бегают часто)
- Статика рендерится в render-texture, динамика - каждый фрейм

Рендеринг: общие принципы

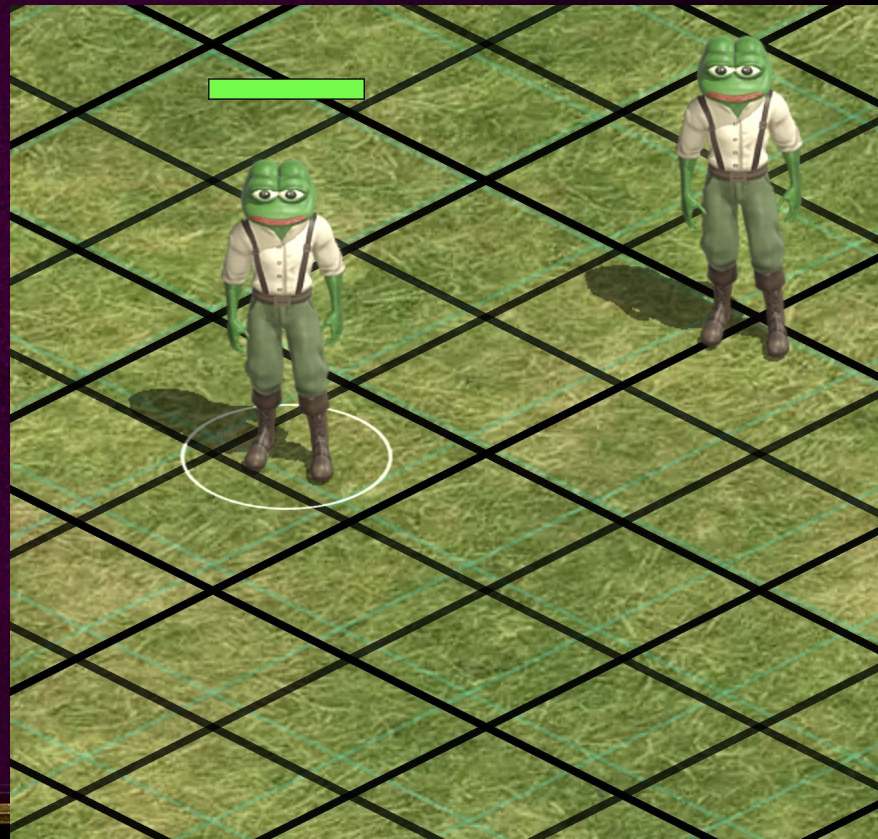
- Везде где можно - GPU Instancing (за основу взяли Pixi Candles)
- Свои culling-слои + вырезаем updateTransform и контейнеры где только можно

Рендеринг: куллинг

- Куллинг - это когда мы рендерим только что, что сейчас на экране
- Главная сложность - посчитать быстро, а что, собственно, на экране

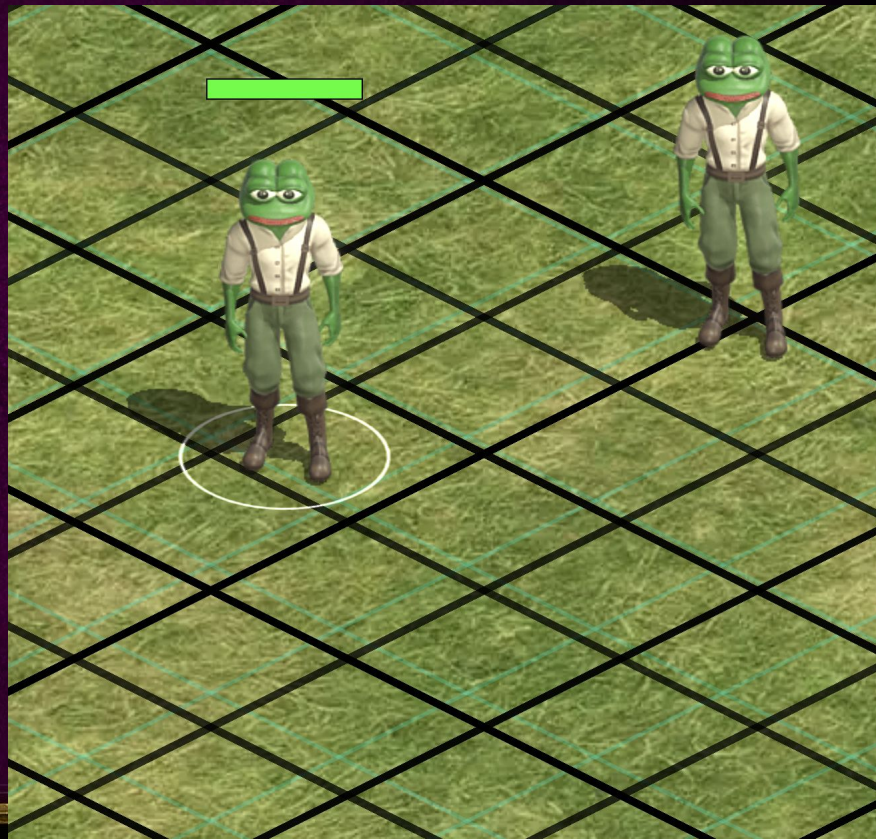
B-tree-like spatial index

- Не R-tree, потому что апдейты частые, сетка заранее известна, ребаланс не нужен
- Без AABB, потому что быстро считаем какие ячейки пересекают сложные формы



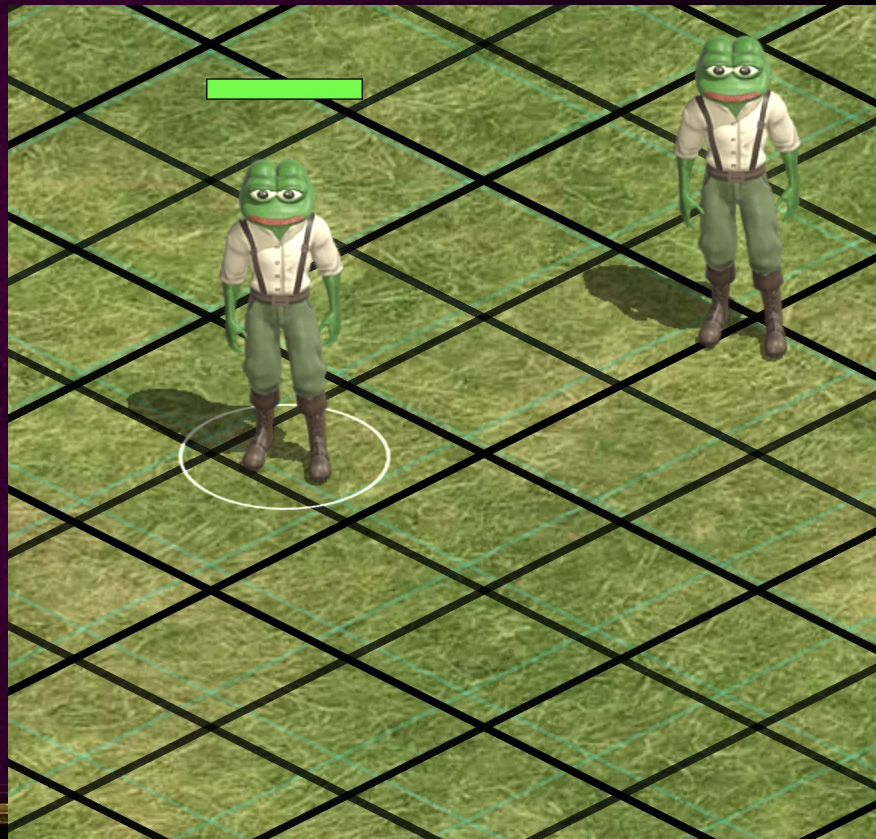
B-tree-like spatial index

- Оптимизация обхода, разделение на moving/static (set/array)
- Запросы в обе стороны - cells by shape, shapes by shape, shape by point



B-tree-like spatial index

- Одна из важнейших структур данных для геймплея, в т.ч. зрения
- В будущем прицепим к нему физику



Режем на слои

- Pixi.js из коробки делает batching для спрайтов, но его можно сломать фильтрами, масками, кастомными объектами между спрайтами

Режем на слои

- У RTS довольно четкие слои высоты - несколько уровней земли (текстуры, индикаторы выделения, etc.), уровни “тел”, уровни “над головой” - health bars, уровни “неба” - туман войны, тень войны

Режем на слои

- Разбивая объекты на неконфликтующие между собой слои получаем отличный батчинг с минимумом усилий
- И тонны updateTransform :)

Свой culling без updateTransform и контейнеров

- Типа pixi-layers, но именно для куллинга

Свой culling без updateTransform и контейнеров

- Есть пространство (игровая карта), в нём объекты (юниты), у объектов есть координаты и форма
- Есть слои, они рендерятся по порядку снизу вверх
- У объекта в слое может быть “представитель” (то что рендерится)

Свой culling без updateTransform и контейнеров

- Используем наш spatial index для мгновенного куллинга объектов
- Когда рендерим слой, то для каждого видимого объекта берём его представителя в данном слое (если есть) и рендерим
- В итоге - updateTransform происходит только для видимых элементов
- А ещё Pixi контейнеры хоть и удобные, но жирные по памяти и чем их меньше - тем лучше

Статика и динамика

- Пример: миникарта или тень войны
- Здания и деревья - статичны, один раз отрендерили и положили в render texture
- Юниты постоянно бегают, рендерим их каждый фрейм

GPU Instancing

- Есть прекрасный pixi-batch-renderer, который умеет делать шаблоны шейдеров и агрегировать текстуры/юниформы в минимальное кол-во batch calls

GPU Instancing

- Но мы хотели супер-легкий инстансинг, вновь же - привязанный к своему куллингу, и без контейнеров (и без Pixi-классов в целом)

GPU Instancing

- Получился небольшой велосипед, который не умеет в textureId/uniformId, но невероятно простой, компактный и быстрый (20 строк для создания слоя с инстансинг-объектами)


AGE
OF
MEMES

Напоследок

Физика в другом потоке

- Физика на замечательном matter-js, всегда на сервере (remote или worker), летит в клиент асинхронно, с тротлингом (чтобы не спамить)

Физика в другом потоке

- Клиент хранит последнее местоположение пришедшее с сервера, данные о пути и скорости
- Клиент считает производное положение второго порядка (считает средневзвешенное между “где должен быть” и “куда бежит”, где вес местоположения пришедшего с сервера падает с течением времени)

Немного о мультиплеере

- Сделали свою абстракцию для сокетов, под капотом либо websocket (remote-сервер), либо channel (worker-сервер)
- Полный гибрид - Node.js сервер и WebWorker-сервер имеют 98% общей кодовой базы, на Node.js только авторизация кастомная

Немного о мультиплеере

- Non-deterministic lockstep с “отражениями” одного и того же компонента на всех клиентах и сервере, и каналом связи между отражениями
- Non-sync state - клиент не получает данные, которые не должен (невидимых юнитов, к примеру)

Немного об архитектуре в целом

- ECS с неэксклюзивными компонентами. В 90% случаев таки эксклюзивные, но если хочется - можно и нет
- Разделение на graphics-тики, state-тики

Спасибо за внимание!

Telegram, неофициальный,
с матом и без цензуры

Twitter, официальный