«Немного» слов о модели игры

(Последняя правка: 28.04.2012)

§1. Настройки мира

Все параметры игры хранятся в файле *config.xml*, который подгружается при создании новой игры.

Файл выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <map>
        <width>100</width>
        <height>100</height>
        // Какие-нибудь ещё константы, отвечающие за генерацию
        // ландшафта
    </map>

    <players>
        <count>5</count>
        // Имена через какой-нибудь разделитель
        <names>Player1 Player2 Player3 Player4</names>
    </players>
</config>
```

Все данные из этого файла парсятся в переменную *properties* типа *WorldProperties*. Структура выглядит так:

```
typedef struct WorldProperties
{
    // World width and height.
    int map_w, map_h;
    // Players' count.
    int players_count;
    // And their names.
    DynArray * player_names;
} WorldProperties;
```

§2. Генерация мира

Сразу же на основании настроек мира строится его «набросок» использованием модуля графа:

- циклический список из игроков (ячейки с типом NODE_PLAYER, рёбра с типом EDGE_NEXT_PLAYER),
- карта (с ландшафтом), тип каждой ячейки NODE_CELL, от каждой ячейки отходят 4 ребра с типами (EDGE_CELL_RIGHT/LEFT/TOP/BOTTOM), т. е. получается четырёхсвязный циклический список.

На этом шаге основная проблема заключается в генерации ландшафта. Используемый пока алгоритм не очень хорош. Подобрать константы генерации тоже не удаётся. Скорее всего, стоит взглянуть в сторону алгоритма шума Перлина.

§3. Юниты

Вся информация хранится в файлах technologies.xml и units.xml. Из них, во время инициализации, подгружаем информацию о юнитах и технологиях.

Для начала расскажу об юнитах, поскольку сама идея выглядит проще и наглядней (для технологий получается аналогично, с небольшими отличиями и наличием дерева).

В файле units.xml информация хранится таким образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<unit>
    <id>1</id>
    <name>Разведчик</name>
    <max_health>10</max_health>
    <max_damage>2</max_damage>
</unit>
```

(Для подробностей см. файл alternative-xml-idea.odt)

Этот файл парсится в такую табличку (динамический массив):

id	name	max_health	max_damage
1	Разведчик	10	2
2	Воин	15	5
3	Мечник	20	10

Таким образом, для *всех* игроков хранится *общая* таблица с информацией о юнитах. Информацию эту можно хранить в виде структуры (а таблицу как массив таких структур):

```
typedef struct UnitCommonInfo
{
    char * name;
    int max_health;
    int max_damage;
} UnitCommonInfo;
```

Для *каждого* игрока также создаётся такой массив:

id	status
1	UNIT_AVAILABLE
2	UNIT_AVAILABLE
3	UNIT_NOT_AVAILABLE

Id тот же, что и в общей таблице. UNIT_AVAILABLE означает, что игрок может нанять этого юнита, UNIT_NOT_AVAILABLE, соотвественно, означает, что игрок не может нанять этого юнита.

Эта информация обновляется после каждого научного исследования игрока (см. ниже).

И, напоследок, имеется такая структура юнита:

```
typedef struct Unit
{
    // id юнита в общеигровой таблице.
    int unit_id;
    // id игрока-владельца.
    int owner_id;
    // координаты x, y в какой-нибудь глобальной системе
    // координат
    int x, y;
    // текущее здоровье юнита
    int health;
} Unit;
```

Они привязаны к карте (своим местоположением) и к своему владельцу. Их атака напрямую зависит от значения health. Можно рассчитывать максимальное возможное значение атаки таким образом: health/max_health * max_damage. Рандом тоже нужно учитывать :)

Каждый юнит помещается в граф. От ячейки, где он сейчас находится, к нему проходит ребро типа EDGE_CELL_UNIT. Чтобы избежать проблем с несколькими юнитами на одной ячейки — мы не будем просто допускать такого, т. е. нельзя будет помещать несколько юнитов на одну ячейку (в точности как в Civilization 5).

Помимо всего прочего юнит помещен в массив DynArray * units в структуре игрока (чтобы игроку можно было легко до него достучаться). Координаты x, y нужны как раз для того, чтобы его быстро найти.

Вот вроде бы и всё с юнитами.

§4. Технологии

С технологиями в общих чертах всё точно так же.

В файле technologies.xml данные хранятся в виде:

```
<?xml version="1.0" encoding="UTF-8"?>
<technology>
    <id>1</id>
    <name>Обработка железа</name>

    <provides>
        <units>2,3</units>
        <technologies>2</techonologies>
    </provides>

    <requires>
        <technologies>0</techonologies>
        <resources>1</resources>
    </requires>
</technology>
```

Файл парсится в дерево (на основе нашего модуля графа), где вершины это технологии (точнее, только их id, ничего больше не надо) типа NODE_TECHNOLOGY, а рёбра типов EDGE_TECH_PROVIDES и EDGE_TECH_REQUIRES.

Такое дерево позволяет легко отслеживать, что требуется для изучения той или иной технологии, и что можно изучить какому-либо игроку.

Создаётся такая табличка (опять-таки *общая* для всех игроков):

id	name
1	Обработка камня
2	Обработка железа
3	Письменность

Эта таблица нужна, чтобы выводить игроку информацию о той или иной технологии (доставать её из дерева, мягко говоря, не удобно).

Для каждого игрока в отдельности создаём такую таблицу:

id	status
1	TECH_RESEARCHED
2	TECH_AVAILABLE
3	TECH_NOT_AVAILABLE

Статус TECH_RESEARCHED означает, что технология уже исследована, TECH_AVAILABLE — доступна для исследования (можно вывести её в каком-нибудь списке в интерфейсе для игрока), TECH_NOT_AVAILABLE — не доступна.

После каждого исследования берём **id** изученой технологии, идём в дерево, смотрим какие технологии позволяет изучать эта технологии, причём не имеют

ли эти самые технологии других неразрешённых «зависимостей» (технологий или ресурсов). Если не имеют — отмечаем в таблице игрока TECH_AVAILABLE для этих технологий. Плюс ко всему, изученная технология может дать доступ к найму новых юнитов, тут обновляется табличка с юнитами.

Поскольку бегать в дерево не удобно, то можно в табличке для всех игроков (где id и name) добавить поле pointer, указывающее на эту технологию в графе.

Остающаяся проблема — это ресурсы, необходимые для изучения технологии. Проблема, на самом деле, решается просто :)

(Я всё ещё придерживаюсь модели: «нельзя изучить что-то, не имея объекта для изучения».)

Создаём для каждого игрока подобный массив (назовём его available_resources):

id	**count**
1	0
2	1
3	3

Id — номер ресурса (unsigned char), **count** — количество найдённых ресурсов *(ресурсы, так-то исчерпаемы, поэтому, например, если у вас 3 ресурса железа, то вы сможете нанять 3 * const мечников)*. Под «найденным» подразумевается, что есть город, находящийся рядом с этим ресурсом.

Тут, правда, возникает другая проблема: какую территорию считать подвластной городу? (И, соотвественно, из каких клеток вокруг города, считать ресурсы принадлежащими ему?) Для нашей простой модели, думаю, хватит какого-то статичного радиуса. Например, все соседние клетки (т. е. квадрат 3 на 3).

Но можно и усложнить модель: ввести влияние населения города на длину этого «радиуса», если на это хватит времени :)

§5. Города

В предыдущих параграфах говорилось (косвенно) о городах. Чтобы достроить модель мира до конца, расскажу и о них.

Структура города должна выглядить таким образом:

```
typedef struct City
{
    // id города.
    int id;
    // id игрока-владельца.
    int owner_id;
    // координаты x, y в какой-нибудь глобальной системе
    // координат
    int x, y;
    // название города
    char * name;
    // население
    int population;
} City;
```

Город даёт такие плюшки:

- позволяет нанимать юнитов, причём, чем больше населения, тем быстрее юниты нанимаются
- увеличивает количество доступных ресурсов
- суммарно у игрока больше населения, значит научные исследования происходят быстрее

Однако у города есть и «негативные» стороны:

- требует денег из казны на содержание (больше населения, больше денег требует)
- город может быть захвачен

В структуре игрока имеется DynArray * cities, с помощью которого он может быстро просматривать список городов.

Население города растёт по какой-нибудь магической формуле, зависящей от возраста города, количества ресурсов рядом и его защищённости (по поводу вопроса о защите городов см. следующий параграф).

§6. Неразрешённые вопросы

1. Нужно ли добавлять зависимость «радиуса» влияния города от населения. Я слабо представляю, как лучше это сделать. Поэтому над этой идеей лучше подумать.
2. Откуда брать игроку деньги? Экономика, только тратящие деньги игрока (пока у нас в модели так) — странная экономика.
3. Как устраивать защиту города? Мне очень нравится идея из Civilization V с самообороной города (там есть очки защиты). Можно её развить и сделать что-нибудь классное.
4. Сколько времени должна изучаться та или иная технология (наниматься тот или иной юнит)? Как решение — задание времени вручную. Есть другое, более элегантное решение — смотреть насколько глубоко технология в дереве (минимальное расстояние от технологии до верха). Чем глубже — тем дольше изучение. Время найма юнита — зависит от технологии, которая позволяет его нанимать.

§7. Easter eggs

При той модели, что у нас есть, можно легко добавлять пасхальные яйца :) И просто необычные для Цивилизации вещи.

Здесь приведены идеи таких пасхальных яиц в модели, которые нужно не забыть добавить (если получится по времени):

- *Летающий город*. Город, который перемещается по карте. Игрок, «поймавший» его (защиты у него никакой) получает золото, бонусы в виде исследованных пары технологий, а также этот самый город. Правда теперь он перемещаться по карте не будет, а будет стоять на том самом месте, где его игрок поймал. Ещё как бонус, можно будет вокруг города сразу после его «поимки» добавить различных редких ресурсов.
- *Червоточины*. Разбросанные по карте червоточины позволят игроку перемещаться быстро из одной точки в другую :) Сделать это просто с помощью добавления такого ребра в граф: EDGE_CELL_WORMHOLE. После исследования какой-нибудь технологии, связанной с ОТО, они становятся видны на карте :)
- *Сундуки, подземелья и проч.* Есть место где разгуляться фантазии.