**Project 4 - диздок**

**Оглавление**

Оглавление

[0. История мира 3](#_Toc437349977)

[1. Геймплей 4](#_Toc437349978)

[2. Действующие лица 5](#_Toc437349979)

[3. Сбор ПЦ 6](#_Toc437349980)

[3.1. Параметры модулей, подсистем и ПЦ 7](#_Toc437349981)

[3.2. Построение ПЦ как модели 17](#_Toc437349982)

[4. Менеджмент ПЦ 19](#_Toc437349983)

[4.1. Переход на экраны менеджмента 19](#_Toc437349984)

[4.2. Экран модулей ПЦ 19](#_Toc437349985)

[4.3. Экран инвентаря 22](#_Toc437349986)

[4.4. Экран юнитов 23](#_Toc437349987)

[4.5. Экран технологий 23](#_Toc437349988)

[4.6. Журнал 23](#_Toc437349989)

[5. Глобальная карта 24](#_Toc437349990)

[5.1. Общий интерфейс 24](#_Toc437349991)

[5.2. Передвижение 25](#_Toc437349992)

[5.3. Действия 26](#_Toc437349993)

[5.4. Контакты и события 26](#_Toc437349994)

[6. Боёвка на тактической карте 28](#_Toc437349995)

[6.1. Генерация поля боя 28](#_Toc437349996)

[6.2. Переход на тактическую карту 30](#_Toc437349997)

[6.3. Ход боя, постройка и поведение юнитов 31](#_Toc437349998)

[6.4. Действия игрока и интерфейс боевого режима 36](#_Toc437349999)

[6.5. Условия завершения боя и итоговый экран 37](#_Toc437350000)

[6.6. Битва более двух ПЦ 37](#_Toc437350001)

[6.7. Атака-бросок 37](#_Toc437350002)

[6.8. Штурм города 38](#_Toc437350003)

[7. Структура и микропроцессы юнита 39](#_Toc437350004)

[7.1. Структура юнита 39](#_Toc437350005)

[7.2. Параметры юнитов 40](#_Toc437350006)

[7.3. Параметры Орудий 43](#_Toc437350007)

[7.4. Стрельба 45](#_Toc437350008)

[7.5. Прицеливание 46](#_Toc437350009)

[7.6. Точность и разброс 51](#_Toc437350010)

[7.7. Анимации 52](#_Toc437350011)

[7.8. Нанесение повреждений 53](#_Toc437350012)

[7.9. Смерть юнита 54](#_Toc437350013)

[7.10. Движение юнита 55](#_Toc437350014)

[8. Способности юнитов 57](#_Toc437350015)

[8.1. Способности юнитов 57](#_Toc437350016)

[8.2. Способности деталей 57](#_Toc437350017)

[8.3. Способности орудий 58](#_Toc437350018)

[8.4. Контрмеры 59](#_Toc437350019)

[9. Мир 61](#_Toc437350020)

[9.1. Генерация мира 61](#_Toc437350021)

[9.2. Репутация игрока 63](#_Toc437350022)

[9.3. Активность фракций 63](#_Toc437350023)

[9.4. … 63](#_Toc437350024)

# История мира

Сеттинг стимпанковый (что представлено технологиями коренного населения мира) и немного магический (что представлено пришельцами из параллельных миров и аномалиями).

Мир представляет собой раздираемый пустотой континент, жители которого цепляются за надежду остановить апокалипсис. Что для этого нужно сделать правда никто не знает, но бороться за любые потенциально ценные артефакты и технологии считается единственно верным решением.

Игрок выступает в роли командующего экспедицией, цель которой спасти мир. Экспедиция размещается в Паровой Цитадели (далее **ПЦ**) – медлительной, но универсальной гигантской машине.

О том, как спасать мир, позднее.

Что особенного в самом мире?

Самую большую ценность представляют Сферы Манипуляции – энергетические сферы/орбы, которые приводят в движение механизмы и позволяют управлять ими на примитивном уровне. Эта энергия способна не только запитать любой двигатель, но и собрать его из готовых деталей по заданной схеме. За считанные секунды.

Многие в этом мире пользуются ими, чтоб собрать, запитать и послать в бой беспилотные боевые машины, называемые мехозаврами – далее **юниты**.

Перед разрушением юнита-носителя, СМ чаще **катапультируется** – после уничтожения они выстреливают СМ вверх и в сторону ПЦ, где их подхватывает магнитный луч, чтобы использовать в следующем мехозавре.

Такая технология дала широкое распространение разным ПЦ – чем содержать и передислоцировать дюжины армий, большинство фракций предпочитают иметь дюжину ПЦ, способных собрать армию перед боем – а при наличии деталей и собрать подкрепления используя СМ подбитых юнитов.

Кроме того, собрать большую армию не позволяет радиационный фон. Если СМ-мехозавров можно собрать, быстро использовать в бою и разобрать после боя, то полноценный танк, который не только воюет, но и путешествует в открытую, нужно оборудовать системами антирадиационной защиты.

Сам мир из-за вторжения пустоты превращается в переплетение нескольких миров (все или большая часть которых пожирается этой самой пустотой). Поэтому в игре кроме коренного населения можно встретить:

1. ХроноЛордов – расу, манипулирующую временем (сеттинг сай-фай/фэнтези)
2. Демонов Пустоты – мистические машины, использующие порталы и свойства пустоты (фэнтези)
3. Морферов – расу живой техники, возникшую из энергии Манипуляции (стимпанк/фэнтези)

Для каждой новой игры генерируется свой набор параметров как условие победы – например, чтобы построить Щит сущности, который останавливает пустоту, в одной игре нужно собрать 7 генераторов антиполя, а в другой 3 генератора антиполя и 5 Нулификаторов. А в третьей игре вообще надо убить Центральный Эгрегор Пустоты.

Поэтому чтобы спасти мир, игрок должен сначала понять, что нужно для спасения мира, а затем реализовать эти условия. Для этого он исследует аномалии/руины/реликты, ищет и собирает различные артефакты, ликвидирует эгрегоры пустоты и её порождения.

Мир очень суров, все отчаянно пытаются выжить, а для этого собрать любой металл, энергию и даже совершенно непонятную хрень – ведь вдруг оно чем-то поможет? А на худой конец можно продать… В общем, это мир выживания, и помогать игроку почти никто не будет.

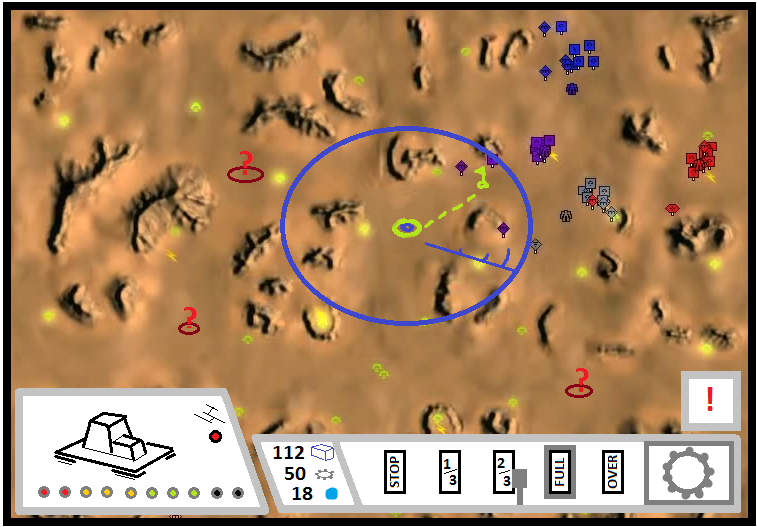
# Геймплей

Игрок принимает роль командира **Паровой Цитадели** (**ПЦ**) – медленной, но универсальной машине, умеющей передвигаться, стрелять, строить, исследовать, добывать ресурсы. Цель: всего-то понять, как спасти мир, и собственно спасти его.

Прямой контроль у игрока будет в основном над ПЦ. Управление в двух режимах – на **глобальной карте** и на **тактической**.

На глобальной карте игрок может:

1. неспешно передвигаться и исследовать разные точки



1. попадать время от времени в ивенты с выбором действий (а ля FTL)
2. при встрече с врагом переходить в боевой режим (на тактическую карту)
3. модернизировать свои параметры и параметры своих войск
4. получать положительную репутацию фракций и находить в них союзников – и отрицательную репу у других фракций, находя в них врагов

На тактической карте (если завязывается бой):

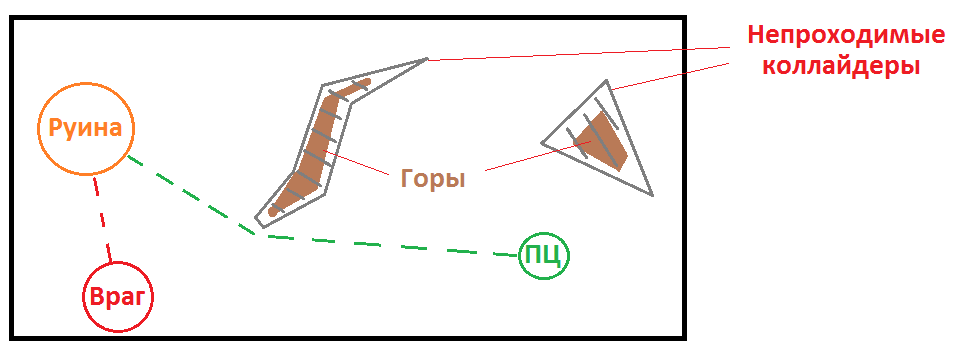
1. игрок строит в ПЦ юниты и указывает им место
2. после сборки юнит выезжает из ПЦ и направляется в это место
3. по достижению этого места юнит направляется в сторону вражеской ПЦ
4. если юнит видит приоритетную цель, он гонится за ней и уничтожает её
5. уничтоженный юнит катапультирует свою СМ, которая возвращается в ПЦ для повторного использования

Бой продолжается, пока на поле не останется лишь одна из сторон – или обе стороны израсходуют ресурсы. Побег возможен, но занимает большое время.

# Действующие лица

Для понимания картины опишем главные действующие лица, составляющие игру. Описания очень краткие – полные описания в соответствующих пунктах.

**Глобал-мап** строится из регионов, которые состоят из лендскейпа (проходимый и непроходимый террейн), а также препятствий и объектов на нём (и те, и другие – меши).



На рисунке **зелёная ПЦ** едет к **оранжевой руине**, огибая **коричневые** **горы** по **серому коллайдеру.** Кроме того, к этой же руине едет **красный враг.** Его коллайдер первым коснётся коллайдера руины, поэтому он первый разыграет тамошний ивент.

Подробнее про глобал-мап [здесь](#_8.1.__Генерация).

**Тактическая карта** представляет собой точно то же, что и глобальная, только имеет ряд дополнительных элементов и несколько новых правил.

Подробнее про тактическую карту [здесь](#_6.1._Генерация_поля).

**Юнит**

Юнит – боевая единица, с которой игрок взаимодействует только на тактических картах.

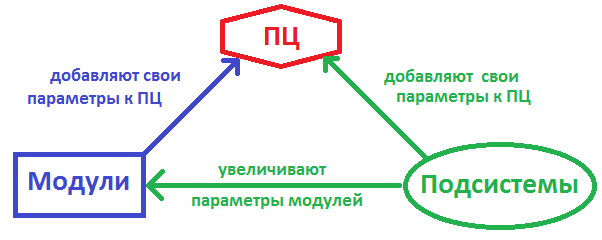
Он самостоятельно передвигается и находит цели (уничтожая врагов или помогая союзникам).

Строится из мешей (FBX-модель). Подробнее про структуру юнита – [здесь](#_Структура_и_микропроцессы).

**Паровая Цитадель**

Основным действующим лицом является ПЦ, и её кратко описывать не будем, так как она описана прямо в следующем разделе.

# Сбор ПЦ

**Для начала пару слов о том, как строится ПЦ.**

ПЦ состоит из модулей, на которые по желанию можно навесить подсистемы.

Предлагается следующая система:

1. ПЦ – это платформа, состоящая из пустых площадок
2. ПЦ сама по себе обладает всеми параметрами (скорость, вместимость и т.д.), но большинство из них равны нулю.
3. В пустые площадки ставятся модули (двигатель, орудия, склад и т.д.) – и они добавляют соответственным параметрам ПЦ свои значения
4. На модули ставятся подсистемы – и они добавляют своему модулю или ПЦ свои значения

Таким образом получим влияние всех модулей на ПЦ, и при этом ПЦ останется одной единицей, с которой удобно работать. Кроме того, при такой системе любой модуль или подсистема смогут добавлять параметры чему угодно – например, двигатель сможет увеличить мощность щита или наоборот.

Чтобы иметь представление о ПЦ, здесь кратко расписаны основные модули:

|  |  |  |
| --- | --- | --- |
| **Название модуля** | **Ставится в площадку** | **Описание** |
| Платформа |  | Не модуль, а вместилище остальных модулей – собственно основа, корпус. |
| Двигатель | двигатель | Модуль, в котором расположен двигатель ПЦ. При поломке Двигателя ПЦ остановится |
| Конструктор | конструктор | Модуль, работающий в основном в бою – он производит детали для юнитов и самих юнитов.  При поломке Конструктора ПЦ останется без возможности пополнять войска в бою |
| Мостик/Контроллер | мостик | Модуль, работающий в основном в бою – он контролирует СМ, вставляемые в юниты.  При поломке Контроллера СМ-юниты начнут воевать хуже – или вовсе не будут. Если у игрока на поле были не СМ-юниты, то их это не коснётся |
| Склад | склад | Модуль, в котором хранятся ресурсы, предметы и вообще всё-всё-всё.  Поломка Склада обычно заключается в разгерметизации отсека и соответственно проникновении радиоактивной пыли.  Заражённые радиацией вещи нужно либо обеззараживать, либо выбрасывать |
| Радар | гипо-площадка | Модуль, отвечающий за обнаружение объектов на расстоянии и возможность с ними поговорить.  При поломке Радара ПЦ остается почти слепой и немой |
| Крупное Орудие | орудие | Крупные Орудия бывают самые разные, но всех их объединяет то, что они работают в бою по приказу игрока (он указал цель, орудие туда бахнуло). |

**Важно:**

На всякий случай сделаем систему, в которой и Модуль, и Площадка могут принадлежать нескольким типам. Площадка может вместить модуль, если хотя бы один её тип совпадает с хотя бы одним типом модуля.

Теперь примеры подсистем, которые можно поставить на модуль:

|  |  |
| --- | --- |
| **Название модуля ПЦ** | **Подсистемы, устанавливаемые на этот модуль** |
| Двигатель | * Продвинутый инжектор (увеличение скорости передвижения) * Контроллер подачи энергии (уменьшает расход топлива) * Укрепление поршней (увеличивает износостойкость) |
| Конструктор | * Дополнительные манипуляторы (увеличивают скорость постройки юнитов) * Магнитный преобразователь (увеличивает скорость создания деталей, используемых для постройки юнитов) |
| Платформа | * Дополнительная секция под модуль (открывает «гипо»-слот для модуля) * Защитная турель (в бою автоматически стреляет при приближении врагов) * Броневой щит (закрывает ПЦ дешёвым в ремонте щитом) |

Кол-во слотов для подсистем у модулей ограничено – по умолчанию 3 слота. Это кол-во можно увеличить, но потолок всё равно есть – по умолчанию 5 слотов.

Слоты для подсистем (в отличие от площадок для модулей) не имеют типов. Зато сами подсистемы ставятся только на те модули, которые перечислены в их ТТХ.

## 3.1. Параметры модулей, подсистем и ПЦ

Перейдём к параметрам МОДУЛЕЙ, из которых собирается ПЦ. Все они выписываются в первую строчку XML-документа, а в первом столбике будут перечислены сами модули.

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | | **Англ.** | **Описание модуля** |
| Рабочее название | | **workname** | Как называется модуль в документации (на это имя ссылаются другие объекты) |
| Игровое название | | **gamename** | Как называется модуль в игре (поле подлежит локализации).  **[TBD]:** Можно здесь использовать ключ, который затем возьмёт данные из отдельного текстового дока |
| Описание | | **description** | Текстовое описание модуля в игре (поле подлежит локализации)  **[TBD]:** Можно использовать ключ |
| Название модели | | **model\_name** | Название FBX-модели, которую использует модуль.  Модель ставится своим пивотом в точку **module\_point** на соответствующей площадке ПЦ.  **[TBD]:** Это поле не нужно, если все модули будут сделаны вручную) |
| Масса | | **mass** | Масса модуля (влияет на скорость ПЦ) |
| Тип модуля:  - конструктор  - двигатель  - хранилище  - мостик  - орудие  - гипо | | module\_type:  - **constructor**  - **engine**  - **storage**  - **bridge**  - **weapon**  - **hypo** | Тип, которому принадлежит модуль.  Проверяется при установке модуля в площадку.  Модуль может принадлежать нескольким типам одновременно (и любая площадка, поддерживающая один из его типов, может принять этот модуль)  Тип модуля также определяет каким образом ему начисляется износ |
| Количество слотов | | * **slot\_amount\_max** * **slot\_amount\_current** | * Количество слотов для подсистем – максимальное * Количество слотов для подсистем – текущее |
| Здоровье | | * **hp\_max** * **hp\_current** | Величина, показывающая структурную целостность модуля:   * максимальное значение * текущее   Если **hp\_current** опускается ниже **0**, модуль переходит в состояние **wreckage**,из которого вариант лишь быть разобранным на металлолом или восстановленным в городе (с некоторой вероятностью) |
| Износ | | **exhaust** | Величина от **0** до **1**, показывающая насколько модуль поломан:  - по накоплению определённого кол-ва износа, может возникнуть поломка  - здоровье модуля/подсистемы не может быть восстановлено ремонтом выше  Каждый раз, когда опускается **hp\_current**, износ модуля/подсистемы повышается на половину потерянного процента здоровья.  *Например, получили* **70** *урона, что составило* **0,12** *от* **hp\_max**  *Значит,* **exhaust** *этого модуля увеличился на* **0,06**  Износ также повышается при использовании модуля *(строительство юнита Конструктором, выстрел из Орудия)*.При использовании модуля износ подсистем растёт вместе с ним |
| Пороговые значения  и вероятность поломки | | **malfunction\_ \_threshold**:  (0.2; 0.3)  (0.5; 0.6)  (0.8; 0.9)  - … | Принимает пары значений – оба от **0** до **1**.  Первое значение – износ **exhaust**, при достижении которого может возникнуть **поломка.**  Второе значение – вероятность наступления поломки на этом пороге.  *Например, имеем* **malfunction\_threshold:**   * *(0.2; 0,3)* * *(0.5; 0,6)* * *(0.8; 0,9)*   *Когда текущий* **exhaust** *модуля поднимается до 0.2, проводится проверка на поломку – на данном пороге вероятность её возникновения 30%.*  *Независимо от результата проверки при повышении износа модуля до следующего порога (0.5) проводится ещё одна проверка – на этот раз вероятность поломки 60%.*  *И т.д.*  **Важно:** когда износ *опускается* до порогового значения (ремонт), проверка на поломку не проводится  Параметр **malfunction\_threshold** может иметь сколько угодно пар значений (порогов) |
| Изнашиваемость | | **exhaust\_speed** | Параметр, показывающий насколько быстро накапливается износ.  Если модуль/подсистема должен был получить **N** единиц износа, то получает **N\*exhaust\_speed**  По умолчанию равен 1. |
| Отражающая броня:  - физическая  - энергетическая  - тепловая | | reflective armor:  - **ref\_arm\_PHYS**  - **ref\_arm\_EN**  - **ref\_arm\_HEAT** | Способность уменьшать получаемый урон на **абсолютную** величину.  Получаемый урон не может стать меньше 1 hp.  *Например: Dmg\_PHYS=5 будет снижен Ref\_Arm\_PHYS=2 до значения 3 (Dmg: 5-2=3)* |
| Поглощающая броня:  - физическая  - энергетическая  - тепловая | | consuming  armor:  - **con\_arm\_PHYS**  - **con\_arm\_EN**  - **con\_arm\_HEAT** | Способность уменьшать получаемый урон на **относительную** величину.  Получаемый урон не может стать меньше 1 hp  *Например:* *Dmg\_PHYS=5 будет снижен Con\_Arm\_PHYS=0,2 до значения 4 (Dmg: 5\*(1-0.2)=4)* |
| Уязвимость | | **vulnerability** | Мультипликатор входящего урона, по умолчанию равен 1.  Считается *перед* срабатыванием всех бронь |
| Стоимость | | **cost\_metal**  **cost\_energy** | Базовая цена покупки и установки модуля.  Эту цену мастерская (в городе или у бродячего торговца) берёт за основу и преобразует для игрока учитывая:  - скидки или наценки из-за личного отношения к игроку  - потребности в ресурсах  *Самостоятельно игрок моет ставить лишь подсистемы – для установки модулей необходимо оборудование Мастерских* |
| Ограничения покупки | | **purchase\_restrictions** | В этом поле указывается раса, в ассортименте мастерских которой может появиться подсистема.  Пустое поле означает отсутствие ограничений |
| Теперь параметры, которые определяют собственно назначение модулей (многие перечисленные параметры будут нулём для не-целевых модулей) | | | |
| Радарная дальность | **radar\_range** | | Радиус, в пределах которого можно идентифицировать цели и общаться с ними |
| Радарная точность | **radar\_acc** | | Точность идентификации целей – определяет количество информации, доступной об объекте |
| Вместимость | **storage\_max** | | Сколько единиц объёма добавляет модуль к общему инвентарю |
| Тяга | **force** | | Тяга – параметр, который конвертируется в скорость. |
| Потребление | **consumption** | | Количество ресурса «энергия», который тратится на **10** секунд передвижения по глобальной карте |
| Контроль СМ | **control\_max** | | Максимальное кол-во СМ, которые ПЦ может контролировать в один момент времени – т.е. выводить в бой в качестве юнитов |
| Щит | **shield\_max** | | Максимальное кол-во **hp** у щита  Щит – купол, останавливающий снаряды на подлёте, тратя на это свою энергию.  При попадании в щит снаряды детонируют, как если бы попали в цель. Их урон наносится щиту по обычным правилам.  Снаряд, взорвавшийся на щите, наносит **АоЕ** урон по всем юнитам вне щита.  Если снаряд гасит щит (опуская его последние **hp** до нуля), то сначала происходит нанесение урона, а лишь затем гаснет щит – то есть **АоЕ** составляющая снаряда не проходит под щит, а успевает заблокироваться гаснущим щитом.  У щита нет отрицательных хитпоинтов (нет состояния **wreckage**).  Щит взаимодействует только со снарядами |
| Поглощение щита | consuming  shield armor:  - **con\_shield\_arm\_PHYS**  - **con\_shield\_arm\_EN**  - **con\_shield\_arm\_HEAT** | | Процент урона, который щит игнорирует по общим правилам (расписано в разделе про боёвку) |
| Базовая регенерация щита | * **shield\_base\_regen\_ \_online** * **shield\_base\_regen\_ \_offline** | | * Базовое кол-во **hp**, которое щит восстанавливает за секунду во включённом режиме * Базовое кол-во **hp**, которое генератор щита заряжает за секунду при отсутствии самого щита – то есть когда щит был уничтожен   Щиты заряжаются ускоренно в выключенном состоянии, а когда доходят до порогового значения **shield\_threshold**, они включаются и начинают регенерировать нормально.  Это базовая регенерация щита, а финальная получается после умножения на **shield\_power** |
| Мощность щита | **shield\_power** | | Параметр, переводящий умножением  **shield\_base\_regen\_online** и **shield\_base\_regen\_offline**  в **shield\_final\_regen\_online** и **shield\_final\_regen\_offline** |
| Пороговый щит | **shield\_threshold** | | Процент зарядки щита, при котором щит включается, если он был выключен.  Например, щит емкостью в **1000** **hp**  был полностью уничтожен, а пороговое значение **40%**.  До **400** **hp**  щит заряжается со скоростью **shield\_regen\_offline** и является выключенным, а по достижению **400** **hp** щит включается и начинает заряжаться со скоростью **shield\_regen\_online** |
| Количество схематик | **scheme\_amount\_max** | | Отображает кол-во типов юнитов, которых игрок может выводить на поле боя.  Игрок может хранить больше схематик, чем использовать и в экране юнитов выбирать, какие схематики вставить.  Замена схематик занимает время. |
| Скорость постройки  юнитов | **construction\_speed** | | Во сколько раз меняется время постройки юнитов, которые обычно строятся за секунд:  По умолчанию параметр равен **1** |
| Вместимость деталей | * **storage\_parts\_current** * **storage\_parts\_max** | | * Текущее кол-во деталей (для постройки юнитов) * Максимальное кол-во деталей   *Для сборки юнитов на СМ-технологии используются специальные детали – они нестабильные, поэтому их нужно хранить в стабилизирующем контейнере.*  Детали, находящиеся в этом контейнере, НЕ занимают место в инвентаре, но увеличивают массу ПЦ. |
| Скорость конвертации | **conversion\_speed** | | Кол-во деталей в секунду, которое ПЦ получает из металла  *Чтобы изготовить детали, необходимые для постройки СМ-юнитов, модуль «Конструктор» берёт из инвентаря металл, преобразует его в детали и кладёт их в стабилизирующий контейнер.*  *Когда игрок закажет постройку юнита, из этого контейнера «Конструктор» мгновенно возьмёт всю сумму, необходимую для постройки юнита, и начнёт его строить.* |
| Эффективность конвертации | **conversion\_eff** | | Кол-во единиц металла, затрачиваемое на изготовление одной детали.  Этот параметр не влияет на **conversion\_speed** – «Конструктор» как получал **N** деталей в секунду, так и получает. Зато этот параметр уменьшает кол-во металла, затрачиваемое на изготовление этих **N** деталей. |
| Эффективность  обратной конвертации  **[TBD]** | **conversion\_reverse\_ \_eff** | | Кол-во единиц металла, получаемое из обломков стоимостью 1 деталь.  В случае победы ПЦ собирает обломки и разбирает уцелевших юнитов на металл (с разными множителями) |
| Боевой потенциал | **potential** | | Величина, определяющая кто сильнее/опаснее.  Каждый модуль и подсистема даёт вклад в общий боевой потенциал.  NPC сравнивают потенциал игрока со своим, чтобы определить, стоит ли нападать на него. |
| Умножитель  параметров | **Multiply** {**модуль; параметр; процент**} | | Если поле пустое, то ничего не происходит.  Если в этом поле есть комбинация {**модуль; параметр; процент**},  то надо выбрать **модуль**, выбрать его **параметр** и повысить его на **процент**.  *Например, если в этом поле мы видим* {**Engine; engine\_power; 0.2**}*, то надо взять текущую* **Мощность\_Двигателя** *модуля* **Двигатель** *(например, было* **60***) и умножить его на* **1.2** *(получим* **72***)*  **Важно:**  Функция **Multiply** применяется одновременно для всей ПЦ, а не последовательно. Например, если два модуля умножают один и тот же параметр одного и того же модуля на **-0.5**, то параметр уменьшается до нуля, а не до **25**, как было бы в случае последовательного применения функции. |
| Перекачка  параметров | **Drain** {**модуль; параметр; процент**} **to** {**модуль; параметр; процент**} | | Если поле пустое, то ничего не происходит.  Если в этом поле есть комбинация {**модуль; параметр; процент**}to{**модуль; параметр; процент**},  то надо выбрать **модуль**, выбрать его **параметр**, понизить его на **процент**, а затем выбрать **модуль**, выбрать его **параметр** и к нему добавить **N процентов** от высосанной величины  *Например, если у одной подсистемы в этом поле мы видим*  {**Engine; forсe; 0.25**}to{**Shield; shield\_power; 0.05**}  *то надо выбрать модуль* **Двигатель***, понизить его параметр* **Тяга** *на* **25%***, а затем выбрать модуль* **Щит** *и повысить его* **Мощность\_щита** *на* **50%** *выкачанной величины*  *(если Тяга была 60 и из неё высосали 25%, то есть 15, то Тяга понизится до 45, а Мощность щита повысится на 50% от выкачанных 15, то есть на 7.5)*  **Важно:**  Как и функция **Multiply**, функция **Drain** применяется одновременно для всей ПЦ, а не последовательно |
|  |  | |  |

В этой таблице нет целого ряда параметров – тех, которые нужны орудиям или другим особым модулям. Эти параметры предлагается выдавать отдельно, чтобы не захламлять XML-документ.

ПОДСИСТЕМЫ  
Имеют почти такие же параметры, как и модули, с некоторыми дополнениями:

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Англ.** | **Описание подсистемы** |
| Совместимость  вместо  типа модуля | [**compatibility**](http://www.lingvo.ua/ru/Search/Translate/GlossaryItemExtraInfo?text=%d1%81%d0%be%d0%b2%d0%bc%d0%b5%d1%81%d1%82%d0%b8%d0%bc%d0%be%d1%81%d1%82%d1%8c&translation=compatibility&srcLang=ru&destLang=en)  вместо  module\_type | Модули, которым подходит подсистема.  В этом поле перечисляются рабочие названия (**workname**) модулей, на которые подсистему можно поставить.  Кроме того, поле может иметь название типа модуля – в таком случае подсистема устанавливается на любые модули такого типа |
| Условия постройки | **build\_conditions** | Условия, при которых подсистему видно в меню улучшений в [экране модулей](#_6.1.__)  Обычно, подсистему можно построить (т.е. она отображается в меню улучшений) лишь в том случае, если в инвентаре есть нужные предметы |
| Улучшаемость | **upgradeability** | Поле определяет, можно ли улучшить подсистему.  Если поле пустое, то нет.  Если в нём присутствуют названия (**workname**) других подсистем, то при выборе данной подсистемы в [экране модулей](#_6.1.__) появится кнопка [улучшить], которая даст выбор из перечисленных в этом поле подсистем.  При выборе одной из них, текущая подсистема удаляется и на её место ставится выбранная, а из счётчика ресурсов вычитается N ресурсов, где |

Итого, имеем следующие XML-документы:

* Параметры модулей
* Параметры подсистем
* Параметры орудий (описаны [тут](#_6.3._Параметры_юнитов))
* Параметры юнитов (описаны там же)

Модули и подсистемы желательно объединить в один док.

Орудия и юниты тоже желательно объединить.

Теперь в какой последовательности это все собирается в кучу:

1. Сначала подсистемы накачивают модули аддитивно (прямым сложением)
2. Затем подсистемы накачивают модули мультипликативно (умножением параметров на число)
3. Затем подсистемы выкачивают из модулей величины аддитивно (прямым сложением)
4. И наконец подсистемы выкачивают из модулей величины мультипликативно (прямым умножением)
5. После всех этих махинаций ПЦ выводит свои финальные параметры

**[TBD]**

Как определяется порядок вычисления параметров?

Предлагается система приоритетов вычислений через добавление в поле XML-дока команды: **(calc\_priority = N).** Если поле имеет перед своим значением такую конструкцию, то этот параметр подсчитывается тем раньше, чем выше **N**. Если конструкции нет, то параметр вычисляется так, будто приоритет равен **1000**.

Наконец после всего этого можно собирать параметры ПЦ.

Для различия параметров ПЦ, модулей и подсистем предлагается ввести теги:

* SUB\_ - параметр подсистемы
* MOD\_ - параметр модуля
* TOTAL\_ - конечный параметр ПЦ

*Например, возьмём параметр «ёмкость» –* **storage\_max*.*** *В случае подсистемы этот параметр выглядит* **SUB\_storage\_max***, в случае модуля* **MOD\_storage\_max***, а ПЦ пользуется параметром* **TOTAL\_storage\_max***.*

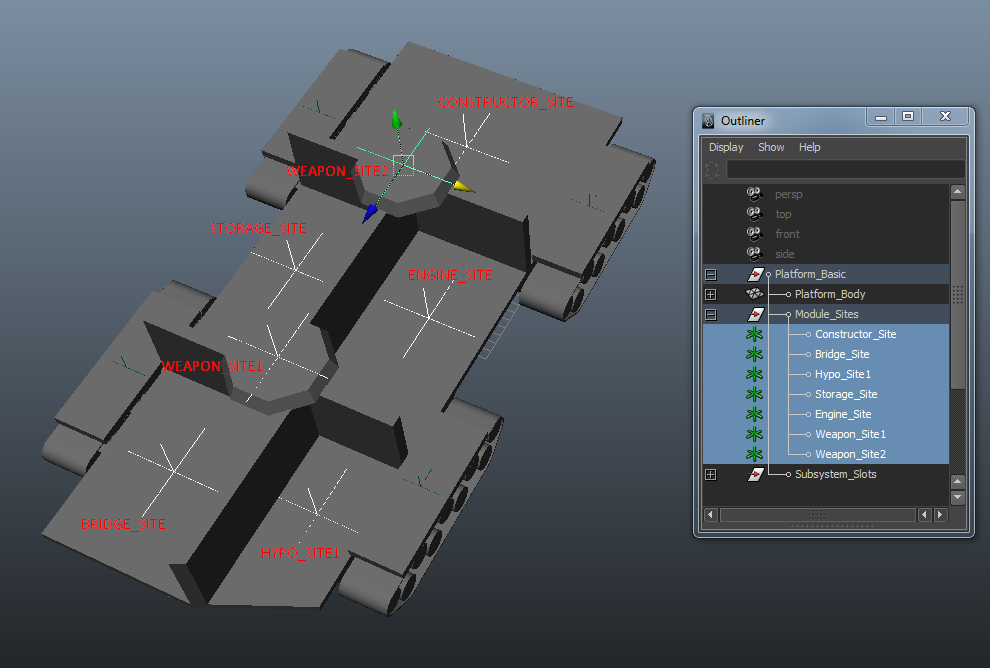
Собственно параметры ПЦ:

Теперь собственно параметры, которые есть у **ПЦ**:

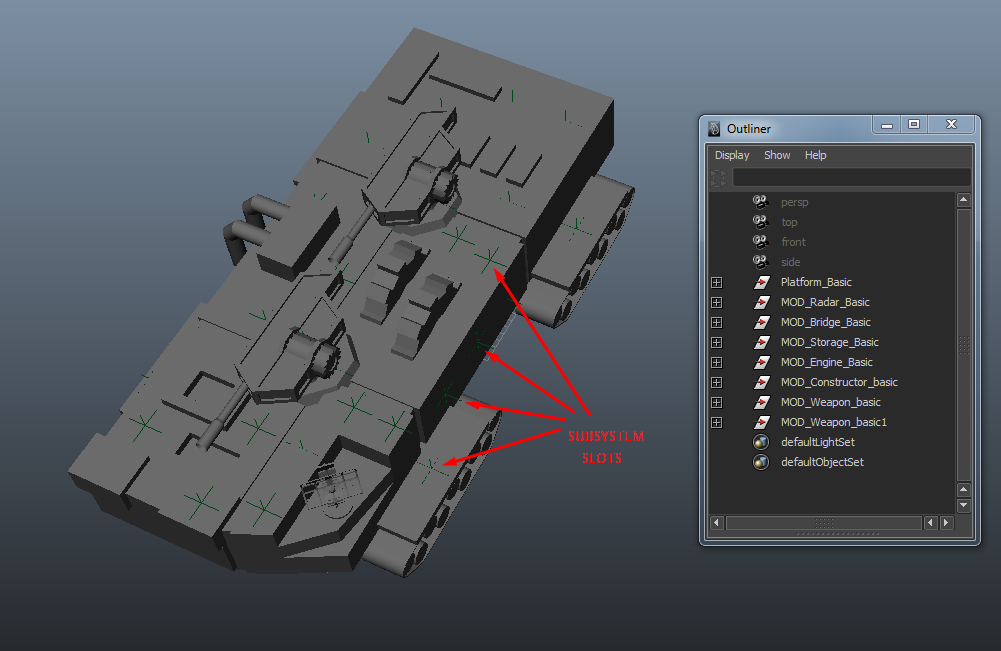
|  |  |  |
| --- | --- | --- |
| **Параметр** | **Англ.** | **Как получить** |
| Текущее и максимальное кол-во модулей  **(вероятно, не нужно)** | * **TOTAL\_amount\_of\_ \_modules\_current** * **TOTAL\_amount\_of\_ \_modules\_max** | * Текущее количество модулей, установленных на ПЦ (двигатель, склад, орудия и т.д.). * Максимальное количество модулей, которые можно установить на ПЦ |
| Загруженность и вместимость | * **TOTAL\_storage\_current** * **TOTAL\_storage\_max** | * Объём ресурсов и предметов, перевозимый ПЦ сейчас * Сумма всех **MOD\_storage\_max** и **SUB\_storage\_max** |
| Радарная дальность | **TOTAL\_radar\_range** | Сумма всех **MOD\_radar\_range** и **SUB\_radar\_range** |
| Радарная точность | **TOTAL\_radar\_acc** | Сумма всех **MOD\_radar\_acc** и  **SUB\_radar\_acc** |
| Масса | **TOTAL\_mass** | Сумма всех **MOD\_mass**, **SUB\_mass**, инвентаря и деталей (детали упомянуты отдельно, т.к. не занимают место в инвентаре) |
| Общая тяга | **TOTAL\_force** | Сумма всех **MOD\_force** и **SUB\_force** |
| Общее потребление | **TOTAL\_consumption** | Сумма всех **MOD\_consumption** и **SUB\_consumption** |
| Скорость глобальная | **TOTAL\_speed\_global** | Скорость, с которой ПЦ передвигается по глобальной карте.  Выводится из параметров **TOTAL\_force** и **TOTAL\_mass** плюс внешние воздействующие факторы (погода, рельеф)  ***A*** и ***B*** – константы, общие для всех ПЦ в игре.  Описано в разделе [Передвижение](#_5.2.__Передвижение) |
| Скорость тактическая | **TOTAL\_ speed\_tactical\_factor** | Множитель, переводящий **TOTAL\_speed\_global** в **TOTAL\_speed\_tactical** |
| Акселерация | **TOTAL\_acceleration** | Быстрота набора скорости на тактической карте.  На глобальной ПЦ набирает скорость мгновенно  ***C*** и ***D*** – константы, общие для всех ПЦ в игре. |
| Скорость поворота  **[TBD]** | **TOTAL\_turn\_speed** | Быстрота поворота на тактической карте.  На глобальной ПЦ поворачивает мгновенно.  Это параметр каждой конкретной ПЦ (точнее модуля Платформа, но покуда он вряд ли будет апгрейдиться, в параметры модулей не введён) |
| Контроль СМ | * **TOTAL\_control\_max** * **TOTAL\_control\_available** | * Макс. контроль СМ равен сумме всех **MOD\_control\_max** и **SUB\_control\_max** * Доступный контроль – максимальный контроль минус используемые в юнитах сферы   Если **TOTAL\_control\_available** равен **0**, то даже при наличии СМ вывести новые юниты в бой нельзя  **Важно:** не СМ-юниты не потребляют контроль |
| Кол-во СМ | **TOTAL\_sm\_amount** | Кол-во СМ, которые имеются в инвентаре и которые соответственно можно использовать для создания юнита |
| Общее кол-во схематик | * **TOTAL\_**   **scheme\_amount\_current**   * **TOTAL\_**   **scheme\_amount\_max** | Кол-во схематик:   * текущее **(вероятно, не нужно)** * максимальное |
| Роль  **(пока не делаем)** | role:  - **SM-control**  - **pilot-control**  - **self-control** | Эта характеристика ПЦ определяет роль ПЦ в бою.  Каждая ПЦ рождается и умирает с одной из трёх ролей:  1) При значении **SM-control** ПЦ контролирует СМ-юниты. Уничтожение такой ПЦ ведёт к выключению её СМ-юнитов в течение 0-5 секунд (они засыпают).  2) При значении **pilot-control** ПЦ командует пилотируемыми юнитами. Уничтожение такой ПЦ не выключает её юниты, а лишь меняет их поведение (дезориентирует).  3) При значении **self-strong** ПЦ командует лишь собой и потому представляет просто крутую боевую единицу. Уничтожение такой ПЦ не меняет ничего, её войска продолжают сражаться как сражались |
| Общий возврат  - металла  - деталей  - энергии  **(вероятно, не нужно)** | - **TOTAL\_recovery\_metal**  - **TOTAL\_recovery\_parts**  - **TOTAL\_recovery\_energy** | Этот параметр показывает, сколько металлолома, деталей и энергии останется после уничтожения ПЦ.  Является суммой параметров **recovery** всех модулей и данной ПЦ с учетом того, живы они или нет (если модуль или подсистема находятся в состоянии **wreckage**, то они вкладывают лишь 30-60% от своего параметра **recovery**) |
| Общий щит | * **TOTAL\_shield\_max** * **TOTAL\_shield\_current** | * Максимальное кол-во **hp** у щита равно сумме всех **MOD\_shield\_max** и **SUB\_shield\_max** * Текущее кол-во **hp** у щита определяется ходом битвы |
| Поглощение щита | consuming  shield armor:  - **TOTAL\_con\_shield\_arm\_PHYS**  - **TOTAL\_con\_shield\_arm\_EN**  - **TOTAL\_con\_shield\_arm\_HEAT** | Сумма соответствующих **MOD\_con\_shield\_arm\_...** и **SUB\_con\_shield\_arm\_...** |
| Мощность щита | **TOTAL\_shield\_power** | Сумма всех **MOD\_shield\_power** и **SUB\_shield\_power** |
| Финальная регенерация щита | * **TOTAL\_**   **shield\_final\_regen\_online**   * **TOTAL\_**   **shield\_final\_regen\_offline** | * Финальное кол-во **hp**, которое щит восстанавливает за секунду при наличии щита: * Финальное кол-во **hp**, которое генератор щита заряжает за секунду при отсутствии самого щита: |
| Пороговый щит | **TOTAL\_shield\_threshold** | Сумма всех **MOD\_shield\_threshold** и **SUB\_shield\_threshold** |
| Скорость постройки  юнитов | **TOTAL\_construction\_speed** | Сумма всех **MOD\_construction\_speed** и **SUB\_construction\_speed** |
| Запас и Вместимость деталей | * **TOTAL\_**   **storage\_parts\_current**   * **TOTAL\_storage\_parts\_max** | * Текущее кол-во деталей (для постройки юнитов) определяется ходом боя * Максимальное кол-во деталей равно сумме всех **MOD\_storage\_parts\_max** и **SUB\_storage\_parts\_max** |
| Скорость конвертации | **TOTAL\_conversion\_speed** | Сумма всех **MOD\_conversion\_speed**  и  **SUB\_conversion\_speed** |
| Эффективность конвертации | **TOTAL\_conversion\_eff** | Сумма всех **MOD\_conversion\_eff**  и  **SUB\_conversion\_eff** |
| Боевой потенциал | **TOTAL\_potential** | Сумма всех **MOD\_potential**  и  **SUB\_potential** |
|  |  |  |
|  |  |  |

## 3.2. Построение ПЦ как модели

Предлагается следующая система сбора ПЦ:

1. Платформа ПЦ представляет собой меш с Площадками – точками для установки Модулей:  
   

Здесь показаны пустые локаторы **Engine\_Site**, **Storage\_Site** и т.д.

1. В эти точки своим пивотом ставятся Модули – отдельные префабы со своими коллайдерами и скриптами. И получаем что-то вроде этого:  
   

Модули пришли со своими слотами для подсистем – **Subsystem\_Slot1 .. Subsystem\_SlotN**.

У ПЦ как платформы есть свои анимации – движение гусениц, их поворот и т.д.

У Модулей свои анимации – движение поршней/шестерёнок и т.д.

Модули – это префабы, у которых своё строение и своя логика. Например, у Модуля Орудие есть группы наведения, точки спауна снарядов и т.д., а также свои скрипты (как целиться, стрелять и т.д.)

# Менеджмент ПЦ

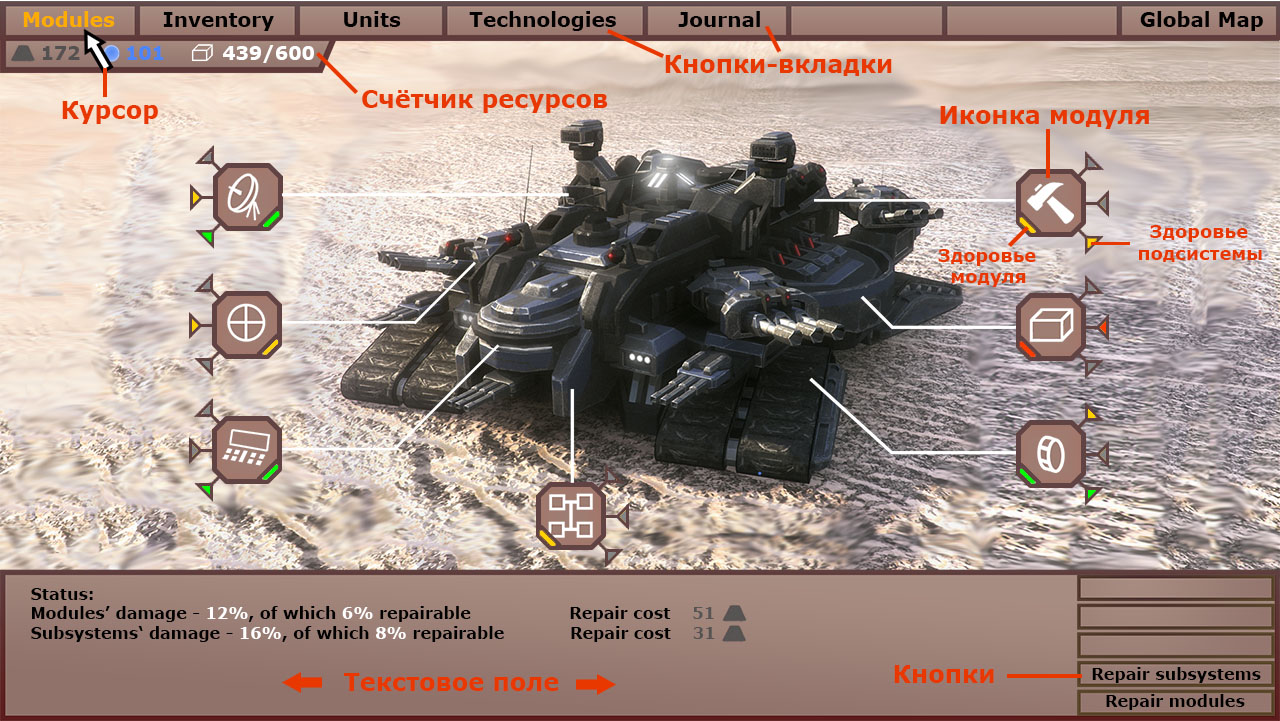
## 4.1. Переход на экраны менеджмента

**Важно:**

Игрок сможет смотреть содержимое не только своей ПЦ, но и вражеских при наличии накачанного модуля Радар.

## 4.2. Экран модулей ПЦ

Первым открывающимся экраном является Экран Модулей:



Здесь:

1. Верх:  
   Вкладки – сейчас выбрана вкладка **[Modules]** (**подсвечивается**).  
   Самая правая вкладка – возврат на глобальную карту – закрывает окно менеджмента
2. Лево-верх:  
   Счётчик ресурсов – металл, энергия, место в инвентаре
3. Центр-низ:  
   Описание общего статуса:  
   - степень повреждений модулей, стоимость ремонта модулей

- степень повреждений подсистем, стоимость ремонта подсистем

1. Право-низ:  
   - кнопка **[Repair Modules]** – изымает стоимость ремонта из счётчика ресурсов и восстанавливает здоровье всех модулей до максимума, сниженного износом  
   - кнопка **[Repair Subsystems]** – изымает стоимость ремонта и полностью восстанавливает здоровье всех подсистема до максимума, сниженного износом

Если ресурсов не хватает, то ремонт не происходит

1. Иконки модулей:

- символ обозначает модуль (на рисунке отмечен Конструктор)

- лампочка в углу отображает здоровье модуля  
 --- зелёная – модуль цел (**hp\_current** >= **90%** от **hp\_max** )  
 --- жёлтая – модуль поврежден, но функционирует (**35-89%**)  
 --- красная – модуль сильно поврежден, но функционирует (**1-34%**)  
 --- красная мигающая – модуль разрушен и не функционирует (находится в состоянии **wreckage**)  
  
- лампочки сбоку отображают наличие и здоровье подсистем:  
 --- серые лампочки означают, что в слоте нет подсистемы  
 --- зелёные – подсистема цела (**hp\_current** >= **90%** от **hp\_max** )  
 --- жёлтые – подсистема повреждена, но функционирует (**35-89%**)  
 --- красные – подсистема сильно повреждена, но функционирует (**1-34%**)  
 --- красные мигающие – подсистема разрушена и не функционирует (находится в состоянии **wreckage**)

По клику на модуль, из него выезжают иконки подсистем:



При этом:

- подсвечивается меш модуля на ПЦ

- иконка модуля с рамкой проигрывает анимацию (увеличивается в размере)

- рамка иконки подсвечивается по краям

- из неё выезжают рамки для иконок подсистем, и лампочки клеятся к ним

- в текстовом поле сверху пишется:  
“Module {**module\_type**} {**gamename**} – {**hp\_current** / (**hp\_max**\*(1 – **exhaust**)) } Repair cost: **N** metal ”

- **hp\_max**\*(1 – **exhaust**) выделено красным и подчёркнуто, а при наведении на это число всплывает хинт

- в текстовом поле посередине появляется текст модуля из поля **description** XML-дока “Modules.xml”

- в самом низу появляются шкалы ненулевых параметров

- внизу справа имеется лишь одна кнопка **[Repair module]**, которая изымает цену ремонта и восстанавливает здоровье модуля до **hp\_max**\*(1 – **exhaust**)

- цена ремонта – 0,3% стоимости модуля за каждый 1% **hp\_max**

Если кликнуть по существующей подсистеме:

- её меш подсвечивается на ПЦ

- рамка иконки подсвечивается по краям

- при этом увеличенный размер иконки модуля остаётся, а подсветка исчезает

- текст кнопки **[Repair module]** меняется на **[Repair subsystem]**

- текст в верху текстового поле меняется на

“Subsystem of {**module\_type**} {**gamename**} – {**hp\_current** / (**hp\_max**\*(1 – **exhaust**)) }  
Repair cost: **N** metal ”

- в текстовом поле посередине появляется текст модуля из поля **description** XML-дока “Modules.xml” (подсистемы входят в тот же док, что и модули)

- ремонт осуществляется по тем же правилам, что и для модулей

Если кликнуть по пустой подсистеме, видим следующее:



- рамка пустого слота подсвечивается

- с верхнего края выезжает менюшка с иконками подсистем, которые можно добавить (рисунку не верим, на нём эта менюшка выехала с левого края)

- кнопки справа снизу становятся пустыми

- текст в текстовом поле меняется на:

“Add new Subsystem to {**module\_type**} Module

Choose Subsystem to see its properties”

И наконец, если кликнуть по подсистеме из списка предлагаемых:



- рамка иконки выбранной подсистемы подсвечивается по краям

- её гипотетический меш на ПЦ начинает мерцать призрачным свечением

- появляются кнопка **[Add Subsystem]** (которая изымает ресурсы и добавляет подсистему в слот)  
и кнопка **[Back]** (которая по нажатию выбирает модуль, на который игрок думал ставить подсистему)

- текст в верху текстового поле меняется на

“Add Subsystem {**gamename**} to {**module\_type**} Module Cost: **N** metal **K** energy”

- в текстовом поле посередине появляется текст модуля из поля **description** XML-дока “Modules.xml”

## 4.3. Экран инвентаря

Свойства предметов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Видимые игроку** | |  | **Невидимые игроку** | |
| **name** | Название (локализуется) |  | **workname** | Рабочее название предмета |
| **description** | Описание (локализуется) |  | **saturating\_rate** | Насколько 1 единица насыщает рынок города |
| **stacks\_till** | Сколько единиц вещи может храниться в 1 слоте *(минимум 1)* |  | **prosperity\_level** | Насколько 1 единица предмета повышает уровень процветания города |
| **mass** | Масса предмета, идёт в счёт **TOTAL\_mass** |  | **military\_level** | Насколько 1 единица предмета повышает уровень военной мощи города |
| **volume** | Объём, занимаемый предметом,  идёт в счёт **TOTAL\_storage\_current** |  | **e-m\_component** | Из скольких электро-магнитных компонентов состоит предмет |
| **metal\_cost** | «Металлическая» составляющая базовой стоимости |  | **hydraulics\_component** | Из скольких гидравлических компонентов состоит предмет |
| **energy\_cost** | «Энергетическая» составляющая базовой стоимости |  | **reinforced\_component** | Из скольких укреплённых компонентов состоит предмет |
| **damage\_level** | Текущий уровень повреждения предмета |  | **void\_component** | Из скольких компонентов «Пустоты» состоит предмет |
| **radiation\_level** | Текущий уровень радиоактивного заражения предмета |  | **chrono\_component** | Из скольких хроно-компонентов состоит предмет |
| **supply\_current** | Текущий уровень заряда/запаса предмета |  | **DLC-0** | Сколько единиц поломки (не **hp**) вмещает Нулевой уровень поломки |
| **supply\_max** | Максимальный уровень заряда/запаса предмета |  | **DLC-1** | Ёмкость 1-го уровня поломки |
|  |  |  | **DLC-2** | Ёмкость 2-го уровня поломки |
|  |  |  | **DLC-3** | Ёмкость 3-го уровня поломки |
|  |  |  | **RLC-0** | Сколько единиц заражения вмещает Нулевой уровень радиоактивного заражения |
|  |  |  | **RLC-1** | Ёмкость 1-го уровня заражения |
|  |  |  | **RLC-2** | Ёмкость 2-го уровня заражения |
|  |  |  | **RLC-3** | Ёмкость 3-го уровня заражения |

Если при добавлении или создании предмета, TOTAL\_storage\_current превышает TOTAL\_storage\_max, то игроку выдаётся сообщение “Storage capacity exceeded – you need to eject something” c кнопкой **[Ok]**. Счётчик места при этом становится красным.

Если игрок пытается вернуться на глобал-мап, то ему снова выдаётся это сообщение, и UI переходит на вкладку Инвентаря.

## 4.4. Экран юнитов

**Параграф в разработке**



Детальное описание процесса апгрейда будет в следующих версиях ГДД.

Кратко:

1. Юнит состоит из мешей, часть из которых изначально отключена
2. На невидимых мешах (или группах мешей) висит скрипт **UpgradeController**.

При апгрейде он включает меш-рендерер, коллайдер и остальные скрипты, висящие на этом меше *(например, если это орудие, то теперь оно отображается и стреляет)*

Чтобы связать визуализацию меша и соответствующий апгрейд, предлагается у скрипта **UpgradeController** сделать опции, доступные в инспекторе:

|  |
| --- |
|  |
|  |

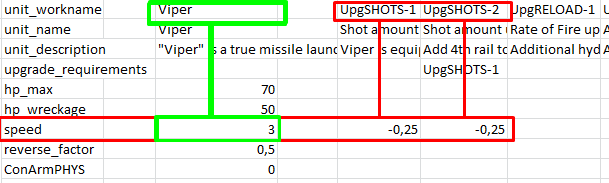
- Начинает отображаться с апгрейда:

- Прекращает отображаться с апгрейда:

Таким образом можно будет подменять улучшаемую деталь на деталь всё лучше и лучше.

1. Характеристики юнита определяются как сумма характеристик его базовой комплектации и всех имеющихся апгрейдов (аналогично ПЦ)

*Например на этом рисунке:*



*Зелёный столбик – характеристики базовой комплектации,*

*Красные столбики – характеристики апгрейдов.*

*Скорость юнита “Viper” в этом примере:*

*- 3 без апгрейдов*

*- 2.75 с одним апгрейдом стволов*

*- 2.5 с двумя апгрейдами стволов*

*Да, скорость становится меньше, потому что доп. стволы много весят. Зато теперь “Viper”стреляет 4-мя, а не 2-мя ракетами (этого таблица не отображает)*

## 4.5. Экран технологий

**Вероятно, функционал будет вырезан**

## 4.6. Журнал

**Параграф в разработке**

# Глобальная карта

Вкратце, глобальная карта (далее **глобал-мап**) создана для путешествия по ней, добычи ресурсов, траты их в магазинах, попадания в события и главное – для расследования дела спасения мира.

На карте есть:

- города (в них надо торговать, ремонтироваться и брать квесты)

- руины (их надо исследовать, частенько попадая в бой)

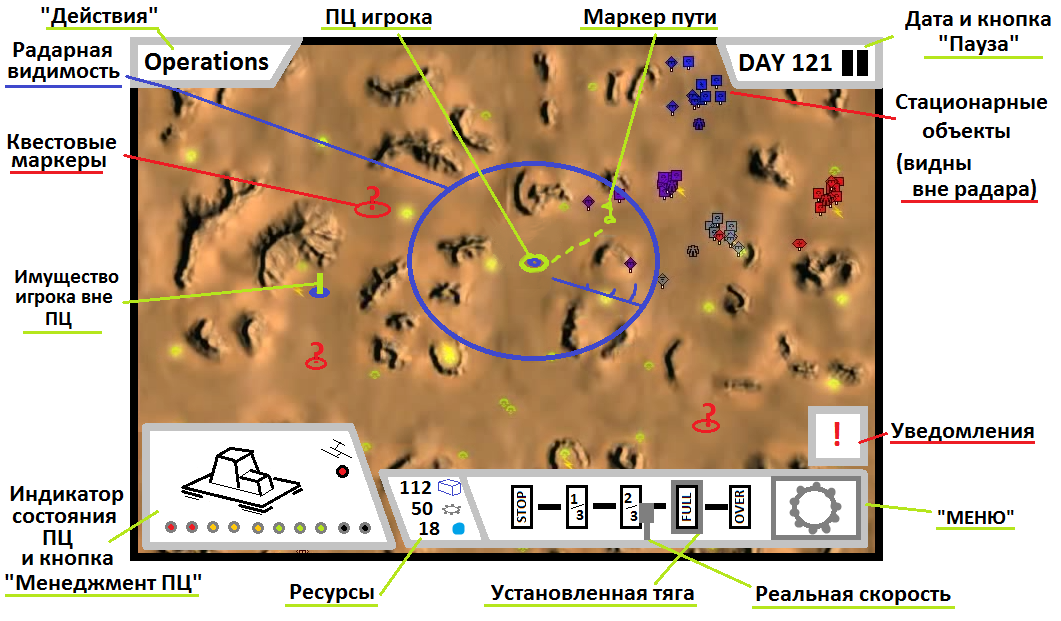
- бродячие NPC (торговцы, боевые части фракций, исследователи, разбойники и т.д.)

- залежи ресурсов (их надо добывать)

Глобал-мап будет выполнен в 3D, но представляет собой плоскость (возможно с мостами). Плоскость проходима везде, кроме препятствий, чья геометрия определяется примитивными коллайдерами.

## **5.1. Общий интерфейс**

В самой первой итерации интерфейс на глобальной карте будет выглядеть так:



Кнопки:

1. Кнопка [Меню] (справа внизу) открывает меню ([Энциклопедия], [Опции], [Сохранить и выйти])

– хот-кеи **“Escape”** и **“F12”**

1. Кнопка [Пауза] (справа вверху) ставит игру на активную паузу (то есть можно отдавать приказы)  
   – на паузу игра ставится и снимается хот-кеями **“space”**, **“p”** и **“pause”.**
2. Кнопка [Менеджмент ПЦ] (слева снизу) ведёт к собственно [экрану менеджмента ПЦ](#_Менеджмент__ПЦ)

– хот-кей **“Tab”**

– ЛКМ по ПЦ также ведёт к экрану Менеджмента

1. Кнопка [Operations] слева-сверху выводит список доступных действий

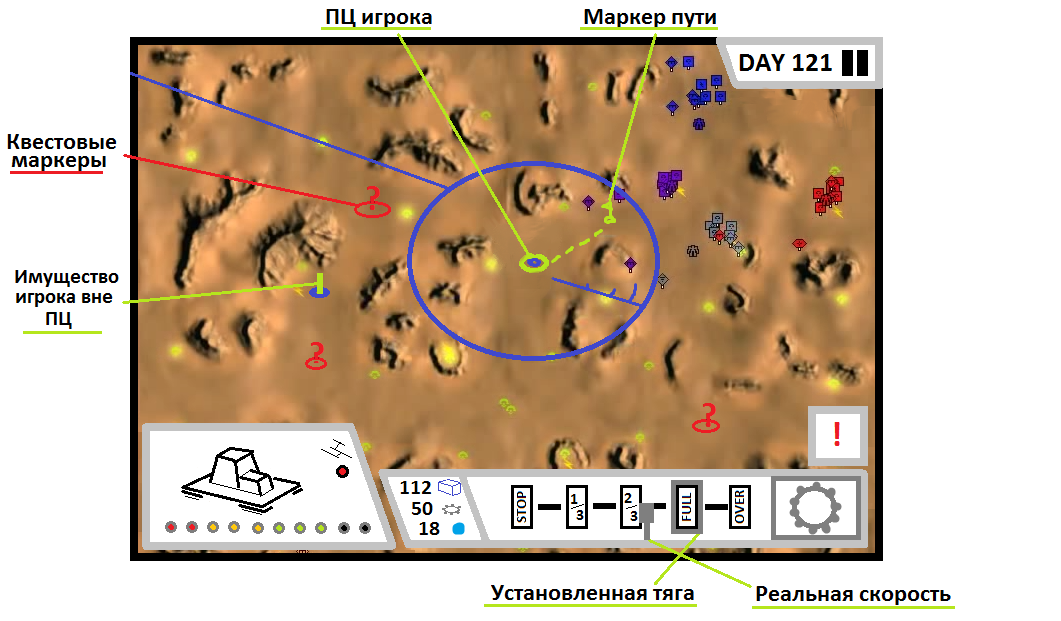
– хот-кей **“Q”** и **"O"**

1. ЛКМ по другим объектам на мапе открывает их диалоговые окна

## 5.2. Передвижение

Передвижение осуществляется так:

1. ПКМ объекту или месту на карте отдаёт приказ ПЦ направиться туда по самому короткому (по времени пути) и безопасному маршруту (ПЦ не выберет маршрут через бурю, пусть он и кратчайший).
2. К этому месту/объекту прокладывается пунктир
3. На само место/объект ставится «маркер пути» – флажок, обозначающий, куда направляется игрок
4. Возле флажка появляется надпись “ETA: {N}”, где N – количество секунд до прибытия.



(скрин устарел: никакой установленной тяги и реальной скорости нет – ПЦ движется всегда в одном режиме)

Развиваемая скорость определяются так:

1. Смотрим параметр Общая Тяга данного ПЦ (**total\_force**, сумма **force** всех модулей и подсистем) при данном режиме движения –
2. Смотрим нынешнюю массу –
3. Вычисляем скорость:

*Например:*

*Тяга, которую выдаёт двигатель, равна 100 – но из-за поломки эффективность двигателя снижена на 20% (получаем 100\*0,8=80).*

*Кроме того, разные подсистемы ПЦ дают ещё 40.*

*Итого:*

*Масса ПЦ игрока – 60 (***total\_mass = 60***).*

*Ставим всё в формулу и получаем:*

Потребление вычисляется по формуле:

(штрафы от поломок уже учтены в )

## 5.3. Действия

## 5.4. Контакты и события

Время от времени игроку взаимодействует с другими представителями мира.

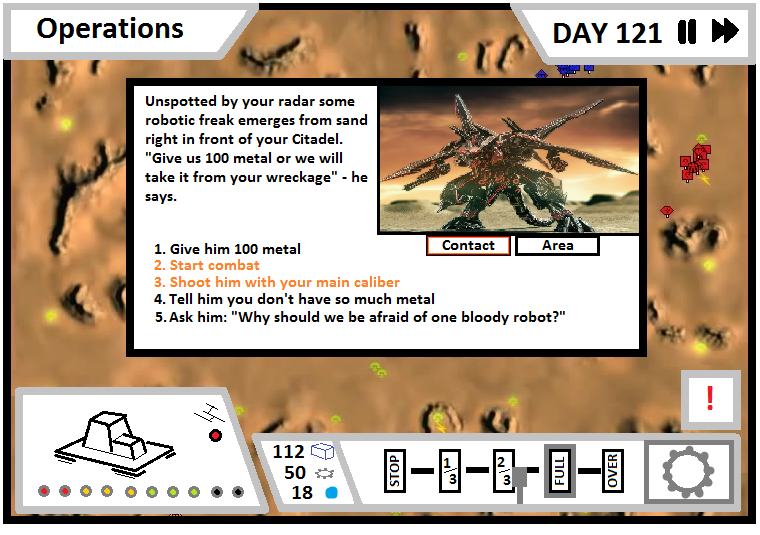
По причине возникновения эти взаимодействия делятся на 2 типа:

1. Контакт с объектом на глобал-мапе (далее **контакт**):  
   - приезд в какой-то город  
   - встреча с другими ПЦ или армиями в пустыне  
   - встреча с торговцами и т.д.
2. Случайное событие (далее **ивент**):

- поломка  
- случайная находка, не обозначенная на карте

- погодное явление и т.д.

Оба типа взаимодействий используют одно и то же диалоговое окно:



- Слева-сверху идёт текст, описывающий контакт/событие

- Снизу даются варианты действий

- Справа-сверху показана картинка

- Под картинкой большинство контактов имеют 2 кнопки – показывать картинку контакта или показывать тактическую карту.

Из этого окна не предполагается выхода иначе, чем как-либо разрешить ситуацию (кликнув по одному из вариантов действий). Однако есть возможность подвигать это окно за рамку, чтоб осмотреть глобальную карту и принять решение с учётом геолокации.

После разрешения контакта или ивента, игрок возвращается на глобал-мап в режиме активной паузы.

Если игрок сам связался с объектом, но передумал что-либо делать, он должен нажать на соответствующую опцию.

Ивент каждого типа имеет свой куллдаун (**КД**). Этот КД никак не зависит от КД ивентов других типов, но возможно будет зависеть.

КД каждого типа ивентов имеет свой диапазон времени.

|  |  |  |
| --- | --- | --- |
| **Тип ивентов** | **Куллдаун** | **Описание** |
| Случайные | 45-75 | Случайное событие, связанное с сеттингом |
| «Поломочные» | 30-45 | Событие, проверяющее поломки на ПЦ |
| Фракционные | 30-45 | Событие, связанное с фракциями, которые имеют вес в данной локации |
|  |  |  |

Как работает куллдаун на примере случайных:  
а) произошёл случайный ивент  
б) рандомизатор выдал случайное значение из диапазона 45-75 минут (локального времени) – получилось 57 мин  
в) через 57 мин рандомизатор бросает кубик на события типа случайный  
г) выпадает «нет события» – игра не встаёт на паузу и никак не оповещает игрока; алгоритм переходит к пункту б)

В отличие от ивентов, которые выпадают самостоятельно, контакты происходят при конкретном стечении обстоятельств – а именно если игрок встретился с объектом на глобал-мапе (с торговцем, с другим ПЦ с городом, руинами и т.д.). Проще всего это реализовать встречей коллайдеров.

Бывают спрятанные ивенты – по сути это контакты, которые не отображаются на глобал-мапе, но которые происходят, когда игрок входит в их триггер-зону.

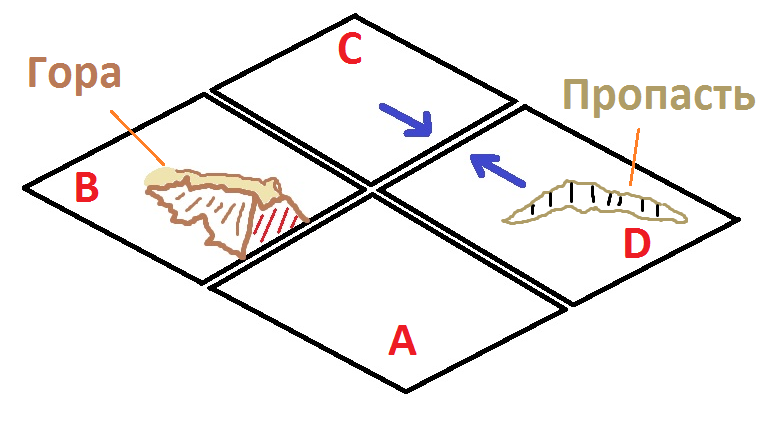
# Боёвка на тактической карте

В этом разделе будет описана боёвка и всё её касающееся.

## 6.1. Генерация поля боя

Предлагается такая система:

Дизайнер локаций готовит квадраты лендскейпа (далее **чанки**), которые перед боем генератор сшиваеи в общую карту:

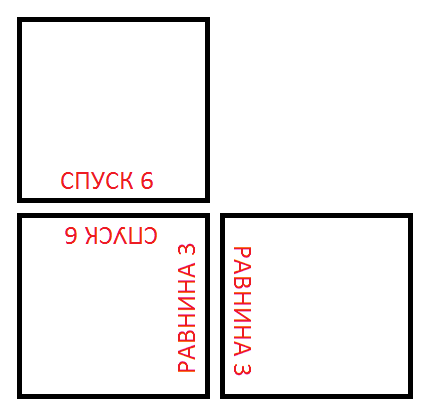


На рисунке:

Посередине чанка “D” пропасть, но она не доходит до края региона, поэтому не мешает стыку. С чанком “B” ситуация другая – гора на нём доходит до края, поэтому его сосед (регион “A”) обязан тоже иметь гору в месте стыка.

Таким образом, предлагается ввести названия сторон чанков, а далее:

1. Первым создаётся угловой чанк
2. К нему ставятся соседние чанки так, чтоб у смежных сторон совпадало название:



И так до тех пор, пока не окажется собранным всё поле.

1. Когда тактическая карта собрана, проводится проверка на проходимость. Если с одного края до другого ПЦ или юниты проехать не могут, то карта собирается снова.

Проходимость проверяется названиями сторон чанков (к примеру, мы знаем, что сторона «обрыв» непроходима).

Если выйдет проверять проходимость по ходу генерации – ещё лучше.

1. Расставляются рандомные препятствия (префабы) в предефайненных точках чанков.

Названия этих точек отображают свойства препятствий, которые они могут там появиться:

- **Small**, **Average**, **Big** – фильтр размера

- **Rock**, **Wrecks**, **Pit**, **Building** – фильтр типа препятствия

- **LP**, **AP**, **HP** – фильтр по вероятности и собственно вероятность появления препятствия в этой точке

- **RHR**, **RVR** (сокращённо от “**Random Horizontal/Vertical Rotation”**) – не фильтр, а указание «можно ли повернуть префаб в этой точке на случайный горизонтальный или вертикальный угол».

Соответственно у префабов-препятствий есть такие свойства:

- Size (**Small, Average, Big**)

- Type (**Rock, Wrecks, Pit, Building**)

- Probability of appearance (**LP, AP, HP**)

- Random rotation (**Horizontal, Vertical**)

Например:

В точках **HP\_Small\_Rock\_Point1 .. HP\_Small\_Rock\_PointN** могут появиться префабы из папки «Rock» со свойствами «высокая вероятность» и «маленький».

Причём вероятность появления такого препятствия в каждой точке высокая.

В инспекторе свойства префаба, который может появиться в этих точках, выглядят так:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Small** | **Average** | **Big** |
| Size | ✓ |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Rock** | **Wrecks** | **Pit** | **Building** |
| Type | ✓ |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **LP** | **AP** | **HP** |
| Probability of appearance |  |  | ✓ |

Поскольку в названии **HP\_Small\_Rock\_PointN** не указано **RVR**/**RHR**, наполнение этих полей

|  |  |  |
| --- | --- | --- |
|  | **Horizontal** | **Vertical** |
| Random rotation | ✓ | ✓ |

… никак не влияет на появление препятствия в точке.

**Важно**:

1. Префаб может иметь несколько параметров одной категории (например, быть и «большим», и «средним») – и может появиться в точке с фильтром, пропускающим любой из этих параметров
2. Одинаковые префабы не могут появиться рядом, даже если обладают разными углами поворота
3. Базовая вероятность появления препятствий в точках меняется местностью глобал-мапы, на которой завязался бой
4. Запекается навмеш, если он используется
5. Если театр боевых действий перемещается к краю поля, то игра ставится на паузу и генерирует новые чанки (игроку выводится текст “Extending battlefield…”)

Технический момент:

Препятствия обладают такими же **hp\_current**, как и юниты – то есть их можно разрушать. Для этого предлагается сделать скрипт **Debris\_Flight**. Он универсальный для анимирования смерти всех объектов из мешей (как препятствий, так и юнитов).

Скрипт вешается на меши или (группы мешей) и при переходе в состояние **wreckage** (то есть когда **hp\_current** опустились до 0 и ниже) включает ригидбоди с какой-то вероятностью.

У скрипта есть 2 параметра:

- вероятность срабатывания

- масса, которая появляется у меша (или группы мешей) с ригидбоди

**Подумать**:

- над многоуровневым разрушением препятствий

- над тем, чтоб остатки разрушенных препятствий выпиливали навмеш, если имеют массу более 0.5 массы ЛТ

## 6.2. Переход на тактическую карту

Бой всегда может начаться в результате **ивента** или **контакта** после нажатия соответствующей опции в **диалоговом окне** (некоторые **контакты** или **ивенты** не имеют других опций).

С ивентами всё ясно, а вот с контактами уточним:

**Контакт** происходит, когда **боевые части** двух сторон сталкиваются на глобальной карте (их коллайдеры пересекаются).

**Боевыми частями** являются **ПЦ** и **армии** (некоторые фракции вместо СМ-юнитов используют пред-построенную и пилотируемую технику, которая объединяется в армию)

Как осуществляется переход в боевой режим:

1. Игрок на глобал-мапе в диалоговом окне нажимает [бой]
2. Время на глобальной карте останавливается
3. Автосохранение игры

- при загрузке этого сохранения игрок сразу попадает в бой

1. Интерфейс режима путешествий уезжает:

- нижняя панель вниз

- верхняя вверх

- не убранные сообщения и индикаторы изготовления предметов – в бок

- непрочитанные сообщения очень желательно сохранить

1. Экран быстро темнеет (желательно с эффектом выключения старого телевизора)
2. Загружается и показывается лоад-скрин боя:  
   - это случайная картинка из папки “Load Screen - Combat”  
   - приезжает снизу прогресс бар
3. Начали загрузку сцены:  
   - генерируется тактическая карта исходя из характеристик текущего райноа глобал-мапы

- загружаются необходимые ассеты (меши, текстуры, скрипты и т.д.)   
- по мере загрузки сцены заполняется прогресс-бар

- по мере по прогресс-бару бегают волны яркости (показывая, что игра не повисла)  
- когда загрузка завершена, прогресс бар уезжает вниз, освобождая мерцающую надпись “Press any key to continue”

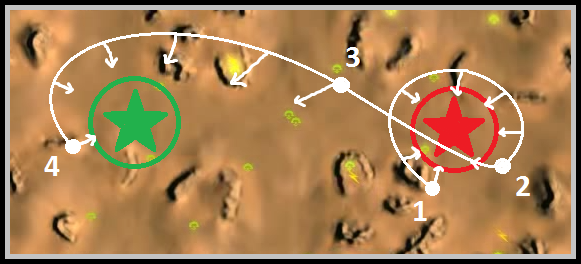
- загрузка 95% или 100% означает, что она реально завершена, а не будет ещё грузиться 100500 минут

1. Игрок нажимает любую кнопку на клаве или мышке
2. Переход на саму карту:

- экран с лоад-скрином быстро темнеет

- включается «полётная» камера с поля боя

- экран светлеет

- начинается полёт «полётной камеры»:  
  
--- камера 5 секунд облетает вражескую ПЦ/армию, держа её в центре кадра (точки 1-2)  
--- затем летит по кривой к ПЦ игрока из точки 2 в 4 примерно 10 секунд  
--- по достижению точки 3 камера плавно поворачивается на ПЦ игрока

--- по достижению точки 4 камера взлетает на стандартную высоту и угол обзора  
(угол поворота камеры можно привязывать не к точкам, а ко времени)

В любой момент игрок может нажать любую клавишу мыши или клавиатуры, чтоб камера быстро взлетела на стандартную высоту.

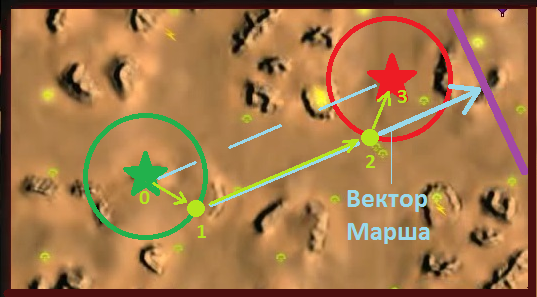
1. Приезжает интерфейс боевого режима
2. Время на глобальной карте начинает идти вновь – со скоростью 0,1 от обычной.

## 6.3. Ход боя, постройка и поведение юнитов

Бой в основном заключается в строительстве юнитов и использовании способностей – их и ПЦ.  
В этом разделе описано общее поведение юнитов.

Большинство юнитов могут пребывать в четырёх основных состояниях:

1. **Деплоинг** – выезд юнита из ПЦ в точку, указанную игроком.
2. **На марше** – поход юнита на ПЦ врага
3. **Штурм** – атака юнитом ПЦ врага
4. **Агр** – атака юнитом других юнитов с уходом с основного маршрута



На данном рисунке:  
**зелёная звезда** – ПЦ игрока,  
**зелёный круг** вокруг неё – зона деплоя игрока,  
**красная звезда** и **красный круг** – ПЦ врага и его зона деплоя  
**голубой пунктир** – направляющая марша (от ПЦ игрока до ПЦ врага),

**голубой вектор** – вектор марша (вектор, из точки деплоя 1 параллельный направляющий),

**фиолетовая линия** – рубеж марша (линия, перпендикулярная направляющей марша и отстоящая от вражеской цитадели на 2 радиуса её деплоя)  
**светло-зелёные стрелки** – маршрут юнита.

Состояния юнита на рисунке:

1. **Деплоинг** (точки 0-1):  
   Игрок приказал задеплоить юнита на самом краю зоны деплоя **(точка 1)** – туда юнит движется из ПЦ игрока **(точка 0)** в состоянии «деплоинг».
2. **На марше** (1-2):  
   После прибытия в **точку 1** юнит переходит в состояние «на марше» – он движется вдоль **вектора марша** на ПЦ врага, объезжая препятствия так, чтоб двигаться максимально близко к **вектору марша**
3. **Штурм** (2-3):  
   Когда юнит попадает в **зону деплоя врага** **(точка 2)**, он переходит в состояние «штурм» - едет на ПЦ врага **(точка 3)** и бьёт её, пока не останется лишь один из них…
4. **Агр:**  
   На рисунке это состояние не описано

Теперь конкретнее по состояниям юнитов:  
  
**1)** **Деплоинг**

Происходит довольно просто:  
*а)* игрок выбирает тип юнита (из панели UI) и кликает по месту в от своей ПЦ – это точка деплоя для данного юнита  
*б)* проверятся текущая **направляющая марша** – вектор от ПЦ игрока к вражеской ПЦ на данный момент (на рисунке выше это **голубой пунктир**)  
*в)* юнит строится N секунд  
*г)* юнит выезжает из ПЦ и направляется в точку деплоя

*д)* юнит строит **вектор марша** – вектор, параллельный **направляющей марша**, берущий начало в точке деплоя и уходящий в бесконечность (можно строить вектор марша на этапе *б)* )

Прибытием в точку деплоя считается сокращение дистанции до неё до **K** метров, то есть вход в её триггер-зону – цилиндрический коллайдер радиусом **K**.

**[TBD]** Если юнит находится в радиусе **2K** более 10 секунд подряд, то считается, что он таки прибыл в точку деплоя (случай, когда внезапно образовавшееся препятствие мешает достигнуть точки деплоя).

Пока юнит находится в состоянии «деплоинг», он на ходу стреляет во врагов, выбирая тех из них, на которых выставлен наивысший **приоритет**:

- Если в **радиусе поражения** появляется цель с **приоритетом**, превышающим приоритет текущей цели **на 2 и больше**, юнит переключается на неё

- После уничтожения текущей цели, юнит переключается на новую цель в **радиусе поражения** с наивысшим **приоритетом**

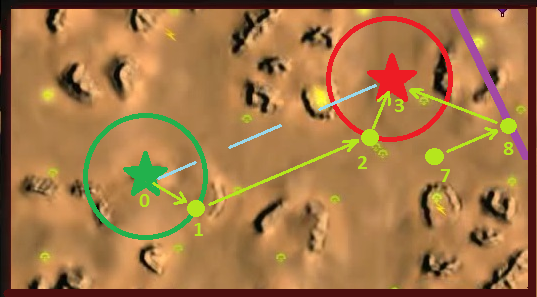
Как только юнит прибыл в точку деплоя, он переходит в состояние «на марше»

**2) На марше**

Юнит двигается вдоль **вектора марша** аж пока не попадёт в **зону деплоя врага**, где он переходит в состояние «штурм».

Может такое случиться, что пока юнит ехал от точки деплоя до ПЦ врага вдоль своего **вектора марша**, ПЦ врага значительно сместилась – и продолжая ехать вдоль **вектора марша**, юнит не попадёт в **зону деплоя врага**. Для этого случая позади ПЦ врага создаётся **рубеж марша** – линия, перпендикулярная **вектору марша** и отстоящая от ПЦ врага на расстояние **2R**, где **R** – радиус **зоны деплоя врага**. Эта линия двигается вместе с ПЦ, но всегда остаётся перпендикулярной **вектору марша**. *(См. рисунок ниже)*

По достижению **рубежа марша**, юнит переходит в состояние «штурм», как если бы приехал в **зону деплоя врага**.



На данном рисунке юнит с маршрутом **0-1-2-3** успел настичь ПЦ врага (попасть в её **зону деплоя**),

а юнит с маршрутом **7-8-3** – не успел, поэтому он доезжает до **рубежа марша (точка 8)**, после чего переходит в состояние «штурм» и сворачивает прямо на ПЦ врага (на **точку 3**).

Как и в состоянии «деплоинг», юнит в состоянии «на марше» на ходу стреляет во врагов, выбирая тех из них, на которых выставлен наивысший **приоритет**:

- Если в **радиусе поражения** появляется цель с **приоритетом**, превышающим приоритет текущей цели **на 2 и больше**, юнит переключается на неё

Однако у состояния «на марше» есть существенное отличие от состояния «деплоинг»:

- Если юнит видит в **радиусе видимости** (не поражения, а именно видимости) цель с **приоритетом** **10+**, он переходит в состояние «агр». Цель с **приоритетом 10+** называется **первичной**

**3) Штурм**

Юнит переходит в это состояние, когда входит в **зону деплоя врага** или пересекает **рубеж марша**.

В этом состоянии юнит подчиняется правилам атаки по **приоритетам** и, как и «на марше», может перейти в состояние «агр».

Единственное отличие состояния «штурм» от состояния «на марше» заключается в том, куда юнит двигается: он старается держаться на расстоянии **0,8 своего радиуса поражения** от ПЦ врага.

Также после уничтожения **первичной** цели юнит возвращается в состояние "штурм", а не в состояние "на марше" (при условии отсутствия других **первичных** целей).

**4) Агр**

В это состояние юнит переходит, если в **радиусе видимости** появился враг с **приоритетом 10 и выше** – то есть **первичный** враг.

Сагренный юнит сворачивает на свою цель и сокращает дистанцию до него до **оптимальной дистанции**.

- Если в **радиус видимости** сагренного юнита попадает ещё новая цель и её **приоритет** выше **приоритета** текущей на **2 и более**, то юнит агрится на новую цель

- Пока юнит сагрен на свою цель, но не достреливает до неё (цель в **радиусе видимости**, но вне **радиуса поражения**), юнит продолжает стрелять во вторичных врагов на ходу, выбирая из них наивысший **приоритет** (как всегда меняет вторичную цель, если её **приоритет** на 2+ больше текущей)

- **[TBD]** Если цель покидает **радиус видимости** (или погибает), то юнит забывает про неё и возвращается в состояние «на марше» (или «штурм», если он в **зоне деплоя врага**) – при условии, что в **радиусе видимости** нет другой **первичной** цели

- Когда юнит из «агра» возвращается в состояние «на марше», он строит новый **вектор марша**: из его текущей позиции параллельный старому (или текущей **направляющей марша** – **[TBD]**).

*Пример:*

*Танк ехал в состоянии «деплоинга» и видел первичную цель (САУ, приоритет 12), но не агрился, поскольку в «деплоинге» агриться неповадно. Так что он просто стрелял в неё и ехал дальше.*

*Как только он доехал до точки своего деплоя, он перешёл в состояние «на марше» – а в этом состоянии агриться можно и нужно, поэтому танк тут же погнался за САУ (других первичных целей в радиусе видимости не было).*

*Как только первичная цель (САУ) была уничтожена, он вернулся в состояние «на марше» – и поехал вдоль вектора марша. Вокруг была одна лишь пехота (приоритет 3), поэтому танк стал на ходу стрелять в неё.*

*Как только в радиусе поражения появился другой танк (приоритет 9), он переключился с пехоты на него, так как приоритет 9 больше, чем 3 на целых 6 единиц (что больше, чем 2 – значит, нужно переключаться).*

*Но тот танк его подстрелил, и потому пример окончен =)*

Некоторые моменты:

- При проверке **радиус видимости** и **радиус поражения** НЕ учитывают, есть ли прямая видимость.

- У многих юнитов имеются второстепенные орудия – их **приоритеты** могут быть совершенно не привязаны к **приоритетам** юнита. Только у главного орудия такие же **приоритеты**, что и у самого юнита (а на самом деле наоборот: это юнит берёт **приоритеты** своего главного орудия)

- Исходя из предыдущего пункта, логичнее сделать так, чтоб сначала агрилось главное орудие – а затем вслед за ним агрился юнит (поворачивал и начинал преследование цели, на которую сагрилось орудие). Но это на усмотрение программиста.

- Если юнит не может атаковать цель, на которую сагрен, в течение 3 секунд (нет видимости или не достреливает), он продолжает преследовать её, но стреляет в другие цели (выбирая наивысший **приоритет**)

- Если юнит не может атаковать цель, на которую сагрен, в течение 12 секунд, он проводит проверку на **первичные** цели снова – и если в **радиусе** **поражения** и в **прямой** **видимости** есть **первичные** цели, он сагрится на ту, у которой наивысший **приоритет**.

- **[TBD, пока не делаем]** Рукопашные юниты вряд ли смогут атаковать «на марше» – они атакуют, в основном только в режиме «агра». Однако если рукопашный юнит не сагрен (находится в состоянии «на марше» или «штурм») и если до вторичного врага рукой подать, то они останавливаются и бьют его, пока тот не уедет от них или не сдохнет. Разумеется, появление **первичной** цели вызывает «агр» по всем правилам.

- Если юниту для атаки нужно перетрансформироваться, то он делает это только в состоянии «агр». Ради **вторичных** целей юнит не трансформируется. Более того, если юнит уже перетрансформирован в боевой режим, а **первичных** целей в **радиусе** **поражения** нет, он трансформируется обратно в походный режим не глядя, есть ли **вторичные** цели.

Хотя возможно, он должен подождать 10 секунд в том случае, если в **радиусе** **видимости** есть приближающаяся **первичная** цель.

## 6.4. Действия игрока и интерфейс боевого режима

Интерфейс боевого режима выглядит так:



Справа снизу – панель юнитов.

Над ней – счётчик доступных СМ:

- Зажжённые лампы соответствуют имеющимся СМ

- если СМ имеется более 10, горит вся шкала

- если меньше 10, то каждой имеющейся СМ соответствует горящая 1 лампа

- точное кол-во СМ (имеющихся/максимальное) указывается справа в цифровом счётчике

- изменения счётчика (и света ламп) происходят сразу, как только началось строительство юнита или как только СМ вернулась в ПЦ

По центру – панель ресурсов:

- слева кол-во металла

- правее (стрелки) индикатор конверсии металла в детали

--- когда синий свет проходит правую стрелку, **conversion\_speed** деталей добавляется к счётчику деталей, а **conversion\_eff** металла убывает из счётчика металла)

--- если в инвентаре нет металла, то стрелки горят красным и детали не производятся

- правее стрелок счётчик деталей

- правее него счётчик энергии

- строительство юнитов сразу изымает полную стоимость из соответствующих счётчиков

Слева – кнопки выбора модулей:

- Крупное Орудие 1

- Крупное Орудие 2

- Инвентарь

- Остальное

Слева-снизу – поле для кнопок способностей выбранного модуля.

Если выбран юнит, то там отображаются кнопки способностей юнита.

По центру снизу – текстовое поле для описания состояния выбранного юнита/ПЦ

Справа сверху **(пока не на рисовано)**:

- кнопка ускорения/замедления времени

- кнопка вызова меню

В бою игрок может совершать следующие действия:

1. Передвигать свою ПЦ с помощью ПКМ:  
   - При нажатии ПКМ по месту, на нём ставится флажок – в этом место ПЦ последует кратчайшим путём и постарается встать на него своим центром

- Если ПКМ пришёлся на непроходимую область, флажок ставится в ближайшее к ПКМ проходимое место

- Если игрок, отдавая приказ на передвижение ПЦ, зажал правую клавишу мыши и двинул её в сторону, то ставится флажок-со-стрелкой – ПЦ приедет на это место и развернётся в направлении, в которое направлена стрелка флажка.

1. Строить юниты:  
   - Игрок выбирает на панели юнита (его иконка загорается)

- Подсвечивается зона деплоя и меняется цвет курсора

- Когда курсор выходит с панели UI, он превращается в призрачный юнит

- Призрак красный вне зоны деплоя и голубой в её пределах

- Когда призрак попадает в зону деплоя и становится голубым, от него прорисовывается стрелка, показывающая **вектор марша**, и текст “ETA: {N}’’ ”, показывающий через сколько секунд юнит построится и приедет в точку деплоя

- Игрок делает ЛКМ, и призрак ставится на точку деплоя, а текст “ETA: {N}’’ ” исчезает

- За 2 секунды до истечения ETA, призрак гаснет

1. Использовать способности орудий и инвентарь **(будет описано позже)**

## 6.5. Условия завершения боя и итоговый экран

## 6.6. Битва более двух ПЦ

## 6.7. Штурм города

# Структура и микропроцессы юнита

Здесь будет описана структура юнита и организация его процессов на микроуровне – макро-поведение описано в предыдущей главе.

## 7.1. Структура юнита

Юнит представляет собой меши и карты: нормал, дифуз, метал/глосс

Модель (формат .FBX) состоит з следующих частей:

1. **{Name}\_Pivot**

- общая группа юнита, определяющая его пивот

*Например, Tornado\_Pivot*

1. **Weapon\_{G}**

- группа орудия {G}

- находится внутри общей группы

*Например, Weapon\_FlameThrowerLeft*

1. **Horizontal\_Rotation\_Group**

- находится внутри группы **Weapon\_{G}**

- поворачивая эту группу наведения вокруг оси OY, юнит целится в горизонтальной плоскости

1. **Vertical\_Rotation\_Group**

- находится внутри **Horizontal\_Rotation\_Group**

- поворачивая эту группу наведения вокруг OX, юнит целится в вертикальной плоскости

- пункты 3) и 4) могут быть поменяны местами (горизонтальная группа может быть внутри вертикальной)

1. **Aim\_Vector**

- пустой локатор

- находится внутри **Vertical\_Rotation\_Group** (или внутри **Horizontal\_Rotation\_Group**, если они поменяны местами)

- юнит производит огонь, когда этот локатор смотрит на цель

1. **Spawn\_Point1..Spawn\_PointN**

- пустые локаторы

- находятся внутри **Vertical\_Rotation\_Group** (или внутри **Horizontal\_Rotation\_Group**, если они поменяны местами)

- в этих точках порождаются снаряды, когда производится выстрел

- снаряды порождаются в порядке возрастания номера группы **Spawn\_Point** (в **Spawn\_Point1** порождается первый снаряд, в **Spawn\_Point2** – второй и т.д.)

- если **Spawn\_Point** с каким-то номером перестал существовать (из-за апгрейда), то выстрел производится из следующего за ним номера

1. **Center\_of\_Mass**- пустой локатор  
   - находится внутри общей группы **{Name}\_Pivot**  
   - означает собой центр масс, который будет использоваться Юнити для реализации физических законов
2. **Wheel\_Colliders**  
   - пустые локаторы  
   - находятся в любых группах **[TBD]**- расположены на осях колёс по середине их ширины  
   - означают точку, в которую становятся колёсные коллайдеры

- называются **WC\_{роль},** где **{роль}**:

--- **motor** – если колесо ведущее

--- **spring** – если колесо подрессоренно

--- **turning** – если колесо поворачивается

- ролей у колеса может быть сколько угодно (минимум одна)

1. **Aim\_Here\_Point [TBD]**

- пустой локатор

- находится внутри общей группы **{Name}\_Pivot**

- означает собой точку, в которую целятся враги, поэтому должна находится там, где попадание в модель наиболее вероятно

1. **Decol\_Point1..Decol\_PointN [TBD]**

- пустые локаторы

- находится в любом месте иерархии модели

- означает собой точку, в которую ставится эмблема/деколь фракции (PNG-файл)  
- скейл **Decol\_PointN** определяет и скейл самой эмблемы, которая туда встанет

- **Decol\_PointN** должны на пару пикселей выше поверхности юнита, чтобы ПНГха не закопалась в меш

Помимо этих обязательных элементов на большинстве моделей будут группы мешей, которые будут отображаться и работать при соответствующем улучшении – их название начинается с **UPG\_...**

Эти меши или группы мешей могут находиться как в **{Name}\_Pivot**, так и в других группах.

Предлагается следующая система работы и отображения таких элементов:

- на FBX-модели изначально присутствуют все меши (лучше текстурить тем же материалом, что и юнит)

- эти меши имеют выключенный меш-рендерер, коллайдер и скрипты их способностей

- на этих мешах висит скрипт **UpgradeController**, который включает меш-рендерер, коллайдер и скрипты способностей этих элементов, если производится соответствующее улучшение

*Например:*

*На танке есть щиты. На них висит скрипт* **UpgradeController** *и* **AbilityArmorShield***.*

*Первый скрипт держит меш-рендерер, коллайдер и второй скрипт выключенными, пока в Экране юнитов не будет произведено соответствующее улучшение схематики.*

*Второй скрипт наделяет меш хит-поинтами, бронёй и способностью быть отстреленным.*

*Когда улучшение будет произведено, на модели начинает отображаться меш щита и включаются его коллайдер и скрипт* **AbilityArmorShield** *– теперь снаряды, летящие в юнит, но угодившие в щит, снижают здоровье щита, а не юнита. Когда здоровье щита снизится до 0, он отвалится и станет декоративным мусором.*

Чтобы связь визуализации меша и соответствующего апгрейда описана [в пункте 4.4.](#_4.4.__)

## 7.2. Параметры юнитов

Нам понадобятся следующие параметры юнитов:

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Англ.** | **Описание** |
| Название юнита:  - художественное  - рабочее | Names:   * **unit\_name** * **unit\_workname** | Названия юнита:  - художественное (поле подлежит локализации)  - рабочее (совпадает с названием в имени XML-файла) |
| Описание | **unit\_description** | Описание юнита (поле подлежит локализации) |
| Хиты здоровья | * **hp\_current** * **hp\_max** | * Текущее кол-во хит-поинтов юнита * Максимальное кол-во хит-поинтов юнита   Когда текущее здоровье юнита опускается до 0 и ниже, он переходит в состояние **“wreckage”**. |
| Хиты обломков | **hp\_wreckage** | * Максимальное кол-во хит-поинтов обломков   Это всегда отрицательное число. Если здоровье юнита находится между этим числом и 0, юнит пребывает в состоянии **“wreckage”**.  Если здоровье опускается ниже этого числа, юнит перестаёт существовать, и его обломки удаляются со сцены. На их месте проигрывается анимация и остаётся мелкий декоративный мусор |
| Стоимость  постройки | **cost\_parts**  **cost\_energy**  **cost\_SM** | Сколько деталей, энергии и СМ нужно потратить, чтобы построить юнит.  Вся стоимость изымается сразу, как только игрок (или ИИ) закажет юнит |
| Базовое время постройки | **basic\_build\_time** | Сколько секунд строится юнит *без* улучшений модуля «Конструктор» |
| Радиус деплоя | **deploy\_radius** | Радиус от ПЦ, в пределах которого можно разворачивать юнит.  Измеряется в радиусах зоны деплоя ПЦ и по умолчанию равен 1.  *Например, значение* **deploy\_radius=2** *будет означать, что юнит берёт текущий радиус зоны деплоя своей ПЦ и умножает его на 2 – в этом радиусе от ПЦ можно развернуть этот юнит.* |
| Скорость глобальная  **(пока не делаем)** | **speed\_global** | Скорость, с которой юнит передвигается по глобальной карте в режиме **long range attack**.  Выводится из **speed\_tactical** c учётом штрафов местности |
| Скорость тактическая | **speed\_tactical** | Быстрота перемены координаты относительно поверхности - возможны подвижные поверхности, относительно которых будет вычисляться скорость  - наклонные поверхности также накладывают штрафы/бонусы (на них будет висеть коллайдер, внутри которого меняется скорость) |
| Фактор реверса | **reverse\_factor** | На этот фактор умножается **speed\_tactical**  при езде задним ходом.  По умолчанию равен 0,5 |
| Скорость поворота | **rotation\_speed** | Быстрота поворота колеса или поворотного механизма |
| Ускорение | **acceleration** | Быстрота набора скорости |
| Масса | **mass** | Масса юнита, когда он жив – то есть не включающая в себя массы мешей со скриптом **Debris\_Flight**.  Желательно массу навесных апгрейдов *(например, броневых щитов)* вынести в отдельно в колонку соответствующего апгрейда (см. [пункт 4.4](#_4.4.__)) |
| Манёвренность  **(пока не делаем)** | maneuverability\_time | Кол-во секунд между попытками юнит повернуть.  Чем меньше это значение, тем чаще юнит будет поворачивать, если находится под огнём противника |
| Возврат | - **recovery\_parts\_ \_factor**  - **recovery\_energy\_ \_factor** | Ресурсы, которые можно собрать после уничтожения/потери данного объекта.  Измеряется в процентах от первоначальной стоимости.  Действует для объектов в состоянии **“wreckage”**.  Если на момент подбора юнит жив, то возврат составляет:  ,  но не более чем  Здесь:  и – общие для всех юнитов множители, зависящие от уровня сложности (на «нормал» равны 1.5 и 0.9 соответственно) |
| Класс | class:  - **infantry**  - **light\_materiel**  - **heavy\_materiel**  - **missile\_launcher**  - **artillery**  - **light\_air**  - **heavy\_air**  - **building**  - **SC** | Класс, которому принадлежит юнит. Может принимать одновременно несколько значений из списка:  - пехота  - лёгкая техника  - тяжелая техника  - ракетница  - артиллерия  - лёгкая авиация  - тяжелая авиация  - здание  - стратегический центр (ПЦ или база)  Если юнит принадлежит сразу нескольким классам, то другие юниты выставляют ему тот наивысший приоритет из тех, что выставлены его классам.  *Например:*  *Юнит «А» принадлежит классам ракетница и тяжелая техника.*  *У вражеского юнита выставлены приоритеты:*  *- тяжелая техника: 9*  *- ракетница: 12*  *Из 9 и 12 враг выбирает наивысший приоритет – 12 (то есть воспринимает «А» как ракетницу).*  *12 ≥ 10, так что юнит переходит в состояние «агр»* |
| Приоритет:  - пехота: X  - лег. техника: Y  - тяж. техника: Z  - и т.д. | priority:  - **infantry**  - **light\_materiel**  - **heavy\_materiel**  - **missile\_launcher**  - **artillery**  - **light\_air**  - **heavy\_air**  - **building**  - **SC** | Насколько важными целями юнит воспринимает другие классы.  Всем существующим классам выставляется приоритет от 1 до бесконечности – и когда юнит встречает их на поле боя, он реагирует на врагов соответственно приоритету, который у него выставлен на их классы  Юнит берёт эти параметры у первичного орудия (описано ниже) |
| Радиус обзора | **sight** | Радиус круга, в пределах которого юнит проверяет наличие первичных врагов.  При наличии таковых прямая видимость не нужна для перехода в «агр» |
| Оптимальная дистанция | **optimal\_distance** | Расстояние, на котором старается держаться юнит от цели |
| Отражающая броня:  - физическая  - энергетическая  - тепловая | reflective armor:  - **ref\_arm\_PHYS**  - **ref\_arm\_EN**  - **ref\_arm\_HEAT** | Способность уменьшать получаемый урон на **абсолютную** величину.  Получаемый урон не может стать меньше 1 hp.  *Например: dmg\_PHYS=5 будет снижен ref\_arm\_PHYS=2 до значения 3 (Dmg: 5-2=3)* |
| Поглощающая броня:  - физическая  - энергетическая  - тепловая | consuming  armor:  - **con\_arm\_PHYS**  - **con\_arm\_EN**  - **con\_arm\_HEAT** | Способность уменьшать получаемый урон на **относительную** величину.  Получаемый урон не может стать меньше 1 hp.  *Например:* *dmg\_PHYS=5 будет снижен con\_arm\_PHYS=0,2 до значения 4 (dmg: 5\*(1-0.2)=4)* |
| Стоимость апгрейда | **upgrade\_cost** | Это поле предназначено для апгрейда юнита, так как они находятся в том же XML-файле.  Измеряется в единицах ресурса «связи».  Когда игрок производит апгрейд, из соответствующего счётчика изымается сумма «связей», указанная в этом поле. |
| Требования к апгрейду | **upgrade\_**  **\_requirements** | Аналогично: это поле предназначено для апгрейда юнита  В этом поле обозначается апгрейд, который нужен для осуществления данного апгрейда.  *Например:*  *У апгрейда UpgSPEED-3 в этом поле требование UpgSPEED-2,*  *у UpgSPEED-2 в этом поле требование UpgSPEED-1,*  *у UpgSPEED-1 это поле пустое.*  *Тогда игрок должен будет сначала провести UpgSPEED-1, что позволит провести UpgSPEED-2, что позволит провести UpgSPEED-3* |
| Стоимость схематики | **scheme\_cost\_metal**  **scheme\_cost\_energy** | Базовая стоимость схематики данного юнита в магазинах |
| Анимации |  | Из этого поля никакой код не берёт никакие данные – тут просто идут описания анимаций и их раскадровки.  *Например:*  *1-180 кадр – движение*  *181-301 кадр – поворот Horizontal\_Rotation\_Group* |

## 7.3. Параметры Орудий

Нам понадобятся следующие параметры орудий:

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Англ.** | **Описание** |
| Хит-поинты  **(пока не делаем)** | **hp** | Очки здоровья объекта. Когда опускается до 0 и ниже – объект переходит в состояние “broken” |
| Скорость поворота  орудия | * **vert\_rotation\_ \_speed** * **hor\_rotation\_ \_speed** | * Быстрота вертикального поворота **Vertical\_Rotation\_Group** относительно локальных координат * Быстрота горизонтального поворота **Horizontal\_Rotation\_Group** относительно локальных координат |
| Зона обстрела | * **vert\_shooting\_ \_zone** * **hor\_shooting\_ \_zone** | * Углы (минимальный и максимальный) относительно локальных координат, в пределах которых может поворачиваться **Vertical\_Rotation\_Group** * Углы (мин. и макс.) относительно локальных координат, в пределах которых может поворачиваться **Horizontal\_Rotation\_Group**   **Примечание:**  Диапазон -180..180 означает, что орудие может вращаться на 360 градусов (оно может перескакивать с -180 на +180, ему не нужно возвращаться в 0) |
| Тип прицеливания:  - настильный  - навесной  - ракетный  - рукопашный | Aim type:  - **trajectory\_flat**  - **trajectory\_plunging**  - **trajectory\_missile**  - **trajectory\_melee** | Какой угол при атаке выбирает орудие.  При расчете угла стрельбы должно появляться два решения – большой угол (навесная или артиллерийская атака)  и маленький (настильная атака, почти прямая наводка)  Типы прицеливания:  - настильный берёт наименьший угол  - навесной берёт наибольший угол  - ракетная берёт наибольший угол  - рукопашная атака не выбирает никакого угла, а просто бьёт;  если коллайдер орудия не встретил коллайдера жертвы, засчитывается промах |
| Скорость снаряда | **muzzle\_velocity** | Скорость полёта снаряда.  Не уменьшается из-за трения о воздух (считаем его нулевым),  но уменьшается при попадании в зону замедления времени  или даже становится отрицательной в зоне инверсии времени  (снаряд летит в обратном направлении)  Эта скорость используется при прицеливании. Модификаторы, которые изменят эту скорость потом, НЕ учитываются.  К параметру **muzzle\_velocity** нужен вдобавок параметр **muzzle\_velocity\_factor,**  который НЕ выносится в инспектор и увеличивает ВСЕ **muzzle\_velocity** в игре:  Этим параметром мы будем настраивать скорость игры |
| Дальность | **range** | Дистанция, в пределах которой орудие может работать  Низкие значения означают рукопашную атаку |
| Радиус повреждения | **aoe** | Радиус, отсчитываемый от эпицентра взрыва и в пределах которого юниты повреждаются |
| Точность | **accuracy** | Вероятность попадания на эталонной дистанции.  Задаётся кол-вом попаданий по эталону, делённым на общее кол-во выстрелов |
| Пиковость  **(пока не делаем)** | **pike** | Определяет выпуклость распределения попаданий |
| Время перезарядки | **reload\_time** | Кол-во секунд между атаками |
| Кол-во выстрелов в атаке | **shots\_amount** | Сколько выстрелов производит орудие за одну атаку (называется **серия**).  Каждый следующий выстрел производится в следующей по номеру **SpawnPointN**  Перезарядка начинается сразу, как только пошёл первый выстрел |
| Время между выстрелами серии | **shots\_time** | Время, за которое выполняются все выстрелы **серии**.  *Например, shots\_time = 1, shots\_amount = 6.*  *Тогда между первым и последним выстрелами проходит секунда – то есть после первого выстрела проходит 1 секунда и остальные 5 выстрелов.*  *В таком случае, между выстрелами будет*  *Выстрелы приходятся на 0; 0,2; 0,4; 0,6; 0,8; 1 сек* |
| Урон:  - физический  - энергетический  - тепловой | Damage:  - **dmg\_PHYS**  - **dmg\_EN**  - **dmg\_HEAT** | Способность уничтожать объекты – эта величина изымается **hp** жертвы (после срабатывания брони и при условии попадания)  Орудие может наносить несколько типов урона сразу  *Например, орудие с dmg\_PHYS=10 и dmg\_HEAT=5 за один выстрел нанесёт 15* ***hp*** *урона (при условии отсутствия соответствующих типов брони)* |
| Пробивание брони:  - физической  - энергетической  - тепловой | - **piercing\_PHYS**  - **piercing\_EN**  - **piercing\_HEAT** | Кол-во единиц брони и её процент, которые будут проигнорированы при попадании. Формула [в главе 7.8.](#_7.8._Нанесение_повреждений) |
| Время зарядки буста | **BoostChargingTime** | Время между нажатием кнопки буста и моментном, когда юнит войдёт в состояние **StateBoostCharged**.  В этом состоянии юнит использует бустер.  *Например:*  *После нажатия на кнопку* ***[Charged Shot]*** *пройдёт* **BoostChargingTime=5** *сек, после чего танк войдёт в состояние* **StateBoostCharged***. В этом состоянии первая атака оглушит цель и получит бонус к точности и энергетическому урону.* |
| Длительность буста | **BoostCharged Duration** | Длительность состояния **StateBoostCharged** при условии, что состояние не пройдёт по другим причинам  *Например:*  *У того же танка состояние* **StateBoostCharged** *длится 6 секунд или пока он выстрелит. Первый же его выстрел получит бонус бустера, но погасит состояние.* |
| Время перезарядки  Буста | **BoostCoolDownTime** | Время между активацией буста и возможностью активировать его снова.  Последовательность: |
| Стэки:  - стана  - перегрева  - дестабилизации | Stacks:  **stack\_stun**  **stack\_overheat**  **stack\_destabilization** | Количество стэков стана/перегрева/дестабилизации, которое приобретает атака юнита в состоянии **StateBoostCharged** |
| Мультипликатор критического урона  **(пока не делаем)** | **crit\_factor** | Показатель, во сколько раз увеличится урон при критическом попадании.  По умолчанию равен 1.5 |
| Шанс критического попадания  **(пока не делаем)** | **crit\_chance** | Вероятность, что попадание нанесёт увеличенный в **Х** раз урон, где **Х =** **crit\_factor** |
| Пристрелка  **(пока не делаем)** | **adjustment** | Количество процентов, которое добавится к **точности** орудия с каждым выстрелом по той же цели |
|  |  |  |

Все данные юнита и его апгрейдов хранятся в одном XML-файле «ТТХ юнита такого-то».

## 7.4. Стрельба

Последовательность действий юнита для стрельбы:

а) В **радиусе поражения** юнита появилась цель – и в силу **приоритетов** юнит решил по ней стрелять

б) Юнит вычисляет угол, по которому надо направить орудие, чтобы учитывая гравитацию снаряд попал в точку, где окажется враг (описано [в след.пункте](#_7.5._Прицеливание))

в) Юнит вращает **Horizontal\_Rotation\_Group** и **Vertical\_Rotation\_Group** так, чтоб **Aim\_Vector** смотрел в небо под тем углом, который был вычислен в пункте б)

- допускается расхождение взгляда **Aim\_Vector** и вычисленной траектории на какие-нибудь 0,1 град, если у системы проблемы с точной наводкой

- если орудие вращалось более чем, скажем, 0.1сек, то после наведения проводится повторное прицеливание

г) Юнит производит выстрелы: в **Spawn\_Point1**..**Spawn\_PointN** проигрывается анимация выстрела и появляются снаряды, которые летят с постоянной скоростью под действием гравитации

- как только пошёл первый выстрел серии, начинается перезарядка

- между выстрелами в одной серии проходит **shots\_time/(shots–1)** – время всех выстрелов поделить на кол-во выстрелов (минус «первый столб», пояснено [в параметрах орудий](#_7.3._Параметры_))

- если юнит или его орудие было уничтожено в середине серии выстрелов, то серия прекращается

- если цель была уничтожена посреди серии, юнит в идеале пытается навестись на следующую цель, но серию не прекращает

## 7.5. Прицеливание

На данный момент предполагается 6 типов прицеливания:

1. Рукопашный  
   - Если цель находится в **радиусе поражения** (для рукопашников это почти в упор), юнит проигрывает анимацию атаки глядя на врага  
   - В конце анимации проводится проверка – если цель осталась в **радиусе поражения**, то ей наносится урон  
   - Если цель пытается уйти от рукопашного юнита, проводящего атаку, этот юнит двигается за ней, стараясь закончить анимацию атаки на враге, а не в воздухе
2. Баллистический настильный с учётом опережения и гравитации  
   - описан отдельно сразу после этого списка
3. Баллистический навесной с учётом опережения и гравитации

- описан вместе с предыдущим пунктом

1. Баллистический с учётом опережения, но без учёта гравитации  
   - описан после баллистического с гравитацией
2. Лучевой  
   - Мгновенное поражение цели с единственным требованием – наличие прямой видимости  
   - Причиняет свой эффект по времени действия на цель (например, наносит 2 hp урона каждые 0,2 секунды, что луч поражает цель)  
   - Вероятно, реализуется банальным рейкастом с визуализацией
3. Ракетный  
   - описан после баллистического без гравитации

**Баллистическая стрельба на опережение с учётом гравитации**

Желающие и имеющие время могут решить систему из трех уравнений движения:

(здесь слева ур-я движения снаряда, а справа – его цели)

А как альтернатива предлагается итерационный алгоритм:

1. *Нулевая итерация:*
2. Юнит целится в цель, как будто та не движется, – в точку
3. Считает, какое время будет лететь снаряд до этой точки
4. *Первая итерация:*
5. Юнит считает, где цель окажется через – точка
6. Считает, сколько времени будет лететь снаряд до точки

Сравниваем и:  
– еслиони отличаются, например, менее чем на 0,05 сек, то принимаем эту итерацию как удовлетворяющую и стреляем по траектории, обеспечивающей время полёта снаряда – если ,то выполняем следующую итерацию

1. *Вторая итерация:*
2. Юнит считает,где цель окажется через – точка
3. Считает, сколько времени будет лететь снаряд до точки

Сравниваем и:  
– если, то принимаем эту итерацию как удовлетворяющую и стреляем по траектории, обеспечивающей время полёта снаряда – если ,то выполняем следующую итерацию.

И так далее, пока не окажется

Как искать ***:***

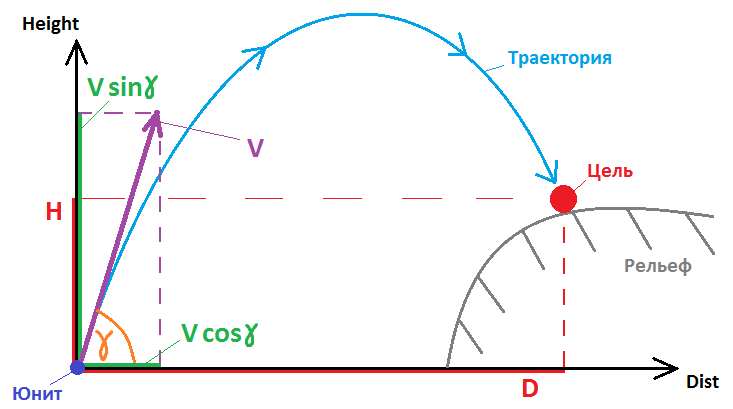
1. Введём величины:

***D*** *–* расстояние до цели по горизонтали

***H*** – расстояние до цели по вертикали (может быть отрицательным)

***V*** *–* скорость снаряда

**𝜸** – зенитный угол, под которым надо запускать снаряд (неизвестен)

**

1. Составим уравнения движения:

Напомним, инаходится с предыдущей итерации – это расстояние по горизонтали и вертикали до точки, где окажется цель через

Важно понимать, что ***OX*** в отличие от ***OY*** не является абсолютной осью – эта ось соединяет стреляющего юнита и точку, в которой цель окажется в данной итерации.

Из первого ур-я: . А из тригонометрии: . Тогда имеем:

Подставляем второе ур-е в первое и получаем:

Бесстрашно умножаем всё на (поскольку ни , ни не могут быть нулём) и раскрываем скобки:

Видим 2 корня для . Но нам нужно само **.**

Берём из последнего выражения квадратный корень и из 2х корней получаем 4 значения ,   
2 из которых отрицательные. Отбрасываем отрицательные как нефизические.

Чаще всего должно получаться, что остаётся два, а не один действительный корень. Один из них будет больше другого – это навесная (артиллерийская) атака. Тот, что меньше, – настильная атака.

Соответственно:

- если орудие имеет тип атаки **навесной**, то для прицеливания оно берёт наибольший угол

- если орудие имеет тип атаки **настильный**, то для прицеливания оно берёт наименьший угол

В итоге для каждого орудия останется лишь один корень.

Когда итерации проведены (каждая следующая не меняет больше, чем на 0.05 сек), это подставляется в уравнение движения по ***OX***:

Отсюда находится угол 𝜸 – зенитный угол, на который должен смотреть **AimVector** орудия, что обеспечивается поворотом **Vertical\_Rotation\_Group**.

Азимутный угол (обеспечиваемый поворотом **Horizontal\_Rotation\_Group**) – это направление ***OX*** (направление на точку встречи, вычисленную в последней итерации).

**Баллистическая стрельба на опережение, но без учёта гравитации**

Во-первых, можно решить систему уравнений:

Здесь слева уравнения движения снаряда, а справа – ур-я движения цели.

Неизвестные: – азимут,– зенит,– время, через которое произойдет встреча снаряда и цели (можно не высчитывать, ведь нужны углы).

Во-вторых, можно использовать предыдущий алгоритм, без члена ☺

**Ракетная стрельба**

Вообще, уравнение движения ракеты примерно такое:

– сила тяги, всегда в направлении движения, постоянная

– коэффициент трения, всегда против направления движения, постоянный

– подъёмная сила, для упрощения всегда направлена вверх, постоянная и равная силе гравитации (но не зависящая от локальной гравитации)

То есть ракета стартует с нулевой скоростью относительно юнита (то есть со скоростью юнита в мировых координатах) и разгоняется до максимальной скорости, ограниченной (скорость, на которой сила трения сравнялась с силой тяги)

**Однако** поскольку такой ракетой будет сложно, предлагается смоделировать ракетную стрельбу примитивно – как стрельбу теми же баллистическими снарядами, только снаряд летит медленней.

Некоторые ракетницы после апгрейда приобретают способность “Homing valley” .

В таком режиме ракеты наводятся:

1. В середине полёта они начинают отклоняться на цель с небольшой угловой скоростью
2. Если цель погибает, то они ищут новую цель среди врагов, ближайших к предыдущей цели.

Вероятность навестись на врага определяется его весом:

Здесь:

– квадрат дистанции от предыдущей цели до ***n***-ного юнита

– масса ***n***-ного юнита

Выбираем цель, на которую навестись:

- суммируем все веса и бросаем кубик на эту сумму

- вес каждого юнита – отдельный бин в этой сумме

- в какой бин попадает значение кубика, на того юнита и наводимся

## 7.6. Точность и разброс

Точность большинства орудий в игре меньше 100% - то есть они промахиваются.

Промахи можно обеспечить двумя способами:

1) снаряд вылетает не точно по **AimVector**'у, а с небольшим отклонением

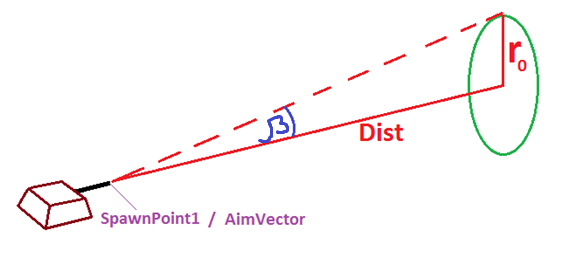
2) количество проводимых итераций при прицеливании ограничено

Второй вариант может давать слабый шанс промаха по целям, которые двигаются навстречу или от стреляющего юнита – но что гораздо хуже, он почти никак не меряется. Поэтому предлагается вариант с отклонением от **AimVector**'а.

Ключевой вопрос: какое отклонение соответствует какой меткости?

Меткость в нашей модели – это процент попадания по эталонной цели на максимальной дистанции атаки. Иными словами это отношение попаданий по эталону к общему количеству выстрелов по ней.

Для понимания предлагаемой модели смотрим рисунок:



**Зелёный** **круг** – эталонная мишень. Она имеет перманентный радиус для всех юнитов, кроме ствольной и реактивной артиллерии – у них этот радиус .

**SpawnPoint1** – точка порождения снаряда.

**AimVector** – вектор, по положению которого орудие понимает, что наведено на цель (на самом деле, он будет внутри орудия).

Отрезок **Dist**, соединяющий орудие и центр мишени, – максимальная дистанция атаки **range**. Для этой дистанции и определяется меткость.

Угол **β** – максимальный угол отклонения, на который снаряд может отклониться от **AimVector** и по-прежнему попасть в цель. Ведь цель не точка, а круг с радиусом. **Пунктир** демонстрирует полёт снаряда по граничной траектории, всё ещё обеспечивающей попадание в круг.

Это значит, что снаряд, отклонившийся от **AimVector на β и** **меньше**, попадает в цель,  
а снаряд, отклонившийся от **AimVector более чем на β**, промахивается.

Ищем **β**:

Значит, для просчёта «попал / не попал» орудие:

1. берёт параметр «дальность» (**range**) из своего [листа характеристик](file:///E:\Z.%20TEMP\5.%20Steam%20Fortress\SvS%20-%20project%204.docx#_7.7._Параметры_орудий)
2. через общий для всех находит **β** (который равен )
3. берёт параметр «меткость» (**accuracy**) из своего листа характеристик
4. и теперь самое сложное – выводится формула вероятности отклонения, по которой:

- **accuracy** процентов должно отклониться не более чем на

- процентов должно отклониться более чем на

- вероятность отклонения понижается пропорционально квадрату отклонения

- функция вероятности непрерывна

Эту формулу предстоит найти разработчикам – вероятно, используя функцию ошибки.

По вышеописанному алгоритму вычисляется отклонение орудий, стреляющих **баллистически без гравитации.**

К сожалению, для **баллистических** **снарядов, которые таки подвержены гравитации**, нужно сделать поправку на кривизну траектории.

Дело в том, что реальная дистанция полёта снаряда, выпущенного по дуге значительно больше дистанции от орудия к эталонной цели. Поэтому для них вычисляется иначе:

, где – длина дуги.

Берём систему уравнений полёта снаряда от орудия до эталонной цели, которая находится на той же высоте, что и орудие:

Длина кривой, по которой летит, вычисляется

либо как

либо как

## 7.7. Анимации

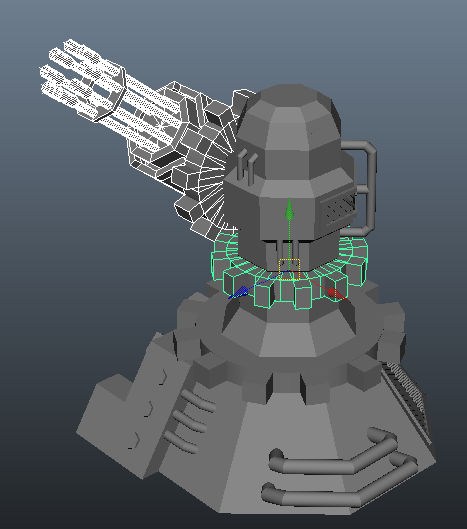
Большинство анимаций предлагается реализовывать в графическом редакторе и подключать их к Юнити через Аниматор-Контроллер.

Основные анимации буду следующие:

1. Движение – анимация циклически проигрывается, пока юнит двигается вперёд, причем скорость проигрывания анимации прямо-пропорциональна скорости юнита  
   *Обычно это движение поршней, шестерёнок, гусениц*
2. Поворот налево/направо – анимация играется с 1-го по последний кадр при повороте в указанную сторону, а по окончанию поворота играется в обратном направлении.  
   Поворот в другую сторону выполняется отзеркаливанием.

*Обычно это поворот колеса и/или поворотного механизма*

1. Атака – анимация начинает проигрываться с 1-го по последний кадр как только произошёл первый выстрел.  
   *Обычно это откат орудий или вращение многоствольного орудия*



1. Вращение **Horizontal-** или **Vertical\_Rotation\_Group** – Юнити берёт тот кадр из анимации, который соответствует данному углу поворота группы наведения.

*Например:*  
*Если Vertical\_Rotation\_Group (на рисунке выделено белым) повёрнута на 41 градус,  
то модель находится в 82-ом кадре этой анимации  
(на рисунке шестерёнка выделена зелёным)  
41-ому градусу соответствует 82-ой кадр, потому что на каждые 0.5 градуса приходится 1 кадр*

1. Трансформация – юнит проигрывает эту анимацию, чтобы поменять режим атаки или приобрести саму возможность атаковать.  
   *Обычно это присуще артиллерии или защитным турелям – которые могут атаковать только из перетрансформированного состояния*

## 7.8. Нанесение повреждений

1. **Посредник**

Не-рукопашным юнитам для нанесения повреждений нужен посредник.

Когда такой юнит атакует, он порождает **болванку**, которая летит со скоростью **muzzle\_velocity** и оставляет за собой короткий шлейф. У некоторых болванок не будет меш-рендерера, а у некоторых – шлейфа.

В случае ракет, в идеале, сделать болванку, ускоряющуюся от 0 до **muzzle\_velocity**.

На болванку действуют все законы игровой физики:

- Гравитация:

Болванку тянет к земле обычная гравитация, что уже было учтено при прицеливании.

Залетая в область аномальной гравитации, болванка следует правилам этой области (например, при антигравитации она начинает взлетать вместо того, чтоб падать)

- Хроно-искажение:

Залетая в область аномального течения времени, болванка следует правилам этой области (снижая свою **muzzle\_velocity** или скорость течения времени – на усмотрение программистов)

- Пространственное искажение:

Залетая в портал-вход, болванка вылетает из парного портала-выхода под углом, под которым влетала в портал-вход.

(Понятия «вход-выход» введены для пояснения – на самом деле, порталы идентичны)

Если портал-выход не параллелен порталу-входу, то болванка, да, меняет траекторию

Болванка взрывается при столкновении с коллайдерами или террейном.

Болванка не имеет принадлежности и потому не различает «свой-чужой».

1. **Попадание и урон**

Когда болванка влетает в коллайдер, она мгновенно убирается со сцены, а на её месте проигрывается анимация её взрыва.

Урон наносится по формуле:

… но не меньше 1 hp.

Индекс суммирования проходит значения от 1 до 3, обозначая типы урона

***Пример:  
Атакующий: dmgPHYS=10, piercingPHYS=0.3 - а остальные составляющие урона нулевые.  
Цель: RefArmPHYS=2, ConArmPHYS=0.5 - а остальные составляющие брони неважны в этом примере.***

***Поскольку урон имеет только физическую составляющую, берём только первое слагаемое – физический урон:***

**[пока не делаем]**

**//** Если вражеский снаряд попадает по мешу из группы «орудие {G}», то урон наносится орудию.

Для начала попробуем модель, в которой попадание по орудию наносит полный урон орудию и половинный юниту, на котором оно стоит.

Если **hp** орудия падает до 0, оно ломается, прекращая стрельбу. Если это было единственное орудие на юните, то тот немедленно самоуничтожается в случае СМ-юнита или едет на ремонт в случае пилотируемого **//**

## 7.9. Смерть юнита

Когда здоровье юнита опускается до 0 и ниже, он:

1. Катапультирует СМ (при наличии таковых) и переходит в состояние **wreckage**
2. Модель «живого» юнита заменяется на модель «мёртвого» юнита (темнее, покарёженнее) в том же кадре анимации, что был «живой» юнит на момент смерти

Группы наведения также повёрнуты так, как были у «живого» юнита

1. Из «мёртвой» модели вылетают те детали, на которых был скрипт **Debris\_Flight**, придающий им ригидбоди и начальную скорость (в направлении, которое указывает локатор **Debris\_Vector**)
2. Из «мёртвой» модели идёт огонь и дым (огонь со временем гаснет, а дым развеивается и перестает идти)
3. Коллайдер у обломков тот, который будет настроен для «мертвого» юнита. По умолчанию он повторяет коллайдер живого юнита.

Напомним, **Debris\_Flight** – скрипт, который вешается на меши или группы и который при переходе в состояние **wreckage** включает ригидбоди с какой-то вероятностью. Получится, что деталь или группа деталей отвалятся или вылетят из юнита – при чём из-за не 100%-ой вероятности каждый раз отваливаться будут разные части.

Если здоровье обломков снижается ниже **hp\_wreckage**, обломки убираются со сцены, разбрасывая декоративный мусор. Коллайдер убирается, и с трупом отныне нельзя делать ничего (воскрешать, разбирать на запчасти).

Юнит, чьё здоровье упало ниже **hp\_wreckage**, идёт в счет уничтоженных врагов, но не даёт ресурсов после битвы.

Катапультирование СМ – это выстреливание СМ в сторону ПЦ так, чтоб она оказалась где-то как-то примерно над ним. Там ПЦ её захватывает магнитными лучами и возвращает к себе назад **(будет описано позже)**.

## 7.10. Движение юнита

Основные моменты движения юнитов:

* Чем физичнее будет реализовано движение, тем лучше
* Юниты объезжают препятствия с небольшим запасом расстояния (во избежание визуальных коллизий)
* Юниты не должны останавливаться при повороте или при перепрокладке маршрута
* Они держатся параллельно поверхности, то есть наклоняют нос на спуске и поднимают его на подъёме.
* Если проход завален уничтожаемыми препятствиями (например, обломками юнитов) и обход завала занимает существенное время, юнит уничтожает обломки
* В идеале реализовать ускорение юнита при разгоне и торможении
* Юниты-не-шагоходы не двигаются боком ни при каких обстоятельствах – помимо редких исключений (когда их притягивают или в случае вертолётов). Юнит двигается вперёд только передом
* Юниты-шагоходы могут двигаться боком, но при этом разворачиваются в направлении движения
* Некоторые юниты не умеют поворачивать стоя на месте – им нужно двигаться.  
  Некоторые не умеют поворачивать на ходу – им нужно остановиться.  
  И наконец, некоторым остановка нужна для стрельбы
* Если мы реализуем реверсное движение, то оно должно происходить со штрафом, указанным в **reverse\_factor**
* Юниты в пере-трансформированном состоянии не могут быть притянуты, но могут быть подняты в воздух.  
  Исключение: здания – их в таком состоянии и в воздух поднять нельзя

# Способности юнитов

## 8.1. Способности юнитов

- Walker:

Юнит ходит по труднопроходимой местности

- Breach (активная способность):

Спустя **T1** сек после активации рукопашный юнит входит в состояние **StateBreachStrike** на **T2** секунд.  
Если в эти **T2** секунд юнит атакует, то получит **K%** буст к скорости на первую атаку и **+K%** урона на неё. Кулдаун **T** сек.

- Self-repair:

Восстанавливает **N hp** каждую секунду.

Способность расходует запас и не может восстановить более **K%** от макс. здоровья за бой.

- Active self-repair (активная способность):

Спустя **T1** сек после активации юнит переходит в состояние **StateActiveSelfRepair:**

засыпает (не делает абсолютно ничего, но по-прежнему подвержен получению повреждений, дебафов и бафов) и восстанавливает **N hp** здоровья каждую секунду на протяжении **T2** секунд.

Куллдаун **T** сек, ограничение – **К** активаций за бой.

- Battery:

Батарея вмещает **N** ед. энергии и восстанавливает **K** ед. в сек.

Предлагается обозначить:

- время зарядки способности **T1** как **BoostChargingTime**,

- длительность буста **T2** как **BoostChargedDuration**,

- время перезарядки способности **T** как **BoostCoolDownTime**

## 8.2. Способности деталей

- Shield:  
Скрипт вешается на деталь, которая загораживает юнита.  
Деталь имеет свой коллайдер, свои хитпоинты и броню. Когда хитпоинты опускаются до нуля, деталь отваливается

## 8.3. Способности орудий

* Stun:

При попадании дальнобойное или рукопашное орудие имеет **K%** шанс наложить на цель **N** стэков дебафа “stun”.

Дебаф “stun”:

* Это каунтер, который уменьшается на **M%** в секунду
* Каунтер повышает значение с получением последующих стэков стана (от любого источника)
* При получении стана цель проверяет, оглушена ли она по формуле
* Если при наложении стэков стана это отношение вышло за 1, то цель оглушается до тех пор, пока не опустит каунтер до 0
* Масса юнита в данных вычислениях – это сумма масс всех его деталей.
* Overheat:

Попадая в цель, орудие имеет **K%** шанс наложить на неё **N** стэков дебафа “overheat”.

Дебаф “overheat”:

* Это каунтер, который уменьшается на **M%** в секунду
* Каунтер повышает значение с получением последующих стэков перегрева (от любого источника)
* Дебаф снижает скорость движения и увеличивает время перезарядки по формуле:

(Здесь функция “signal” отсекает дебаф, если он мал)

* Масса юнита в данных вычислениях – это сумма масс всех его деталей.
* Shredder:

Атака рукопашного юнита или снаряды, выпущенные орудием, навсегда убавляют у цели **N** брони всех типов

Данная способность не может опустить броню ниже нуля.

* Energy attack:

Атака орудия расходует **N** ед. энергии батареи.

Если у юнита нет **N** ед. энергии, то атака данным орудием не проводится, пока юнит не накопит это кол-во энергии.

* Charged Strike (активная способность):

По активации способности орудие (или оружие ближнего боя) заряжается в течение **T1** секунд, после чего входит в состояние **StateBoostCharged** на **T2** секунд.

Следующие **N** атак (по умолчанию **N=1**) в этом состоянии получают **+A** к точности и наносят **+D** энергетического урона и **S** стэков дебафа “stun”.

Когда юнит выстрелит эти **N** атак или когда пройдёт **T2** секунд, юнит получает **+R%** к времени перезарядки на **T3** секунд.

Кулдаун **T** сек, начинает отсчитываться с момента, когда был израсходован последний заряженный снаряд или по прошествии **T2** секунд (если не все выстрелы были израсходованы).

## 8.4. Контрмеры

**(пока не делаем)**

1. Если способность не на перезарядке, то как только в радиусе **Х** метров замечена вражеская ракета, автоматически выпускаются все имеющиеся контр-меры (и начинается перезарядка способности)
2. **N** болванчиков появляются в **N** спаун-точках (назовём их **CounterSpawnPoint1** .. **CounterSpawnPointN**)
3. Каждый болванчик вылетает по вектору своего **CounterSpawnPointN**

Его начальная скорость – рандомная величина из узкого диапазона [**V1**..**V2**] (то есть, контр-меры вылетают с немного отличающимися скоростями)

1. Болванчик летит по траектории, описываемой физическим законом тяготения – т.е. на него действует ускорение **A\*g** (часть ускорения свободного падения)
2. Болванчик «пытается стать целью самонаводящегося снаряда» в радиусе **Y** от себя

- он делает проверку каждый раз, ракета влетает в радиус **Y** (то есть чаще всего у него 1 попытка на 1 ракету)

- он преуспеет с вероятностью **Z%**

Если он преуспел, он прекращает попытки стать целью других ракет.

1. Болванчик выполняет пункт 4) для каждой входящей ракеты, пока не преуспеет или умрёт:

- а умрёт он через **T2** секунд после появления на свет или пока не столкнётся с каким-либо снарядом (даже не ракетой)

- если болванчик сталкивается со снарядом-не-ракетой, он всё равно вызывает детонацию снаряда

1. Пока жив, болванчик представляет собой огонёк с дымовым шлейфом (максимально близкий к реальным контр-мерам)

Когда умер, болванчик не представляет собой ничего (удаляем его со сцены, проигрываем анимацию смерти – слабую-слабую вспышку и угасание дыма)

Таким образом нужны параметры:

**А** - ускорение свободного падения специально для болванчиков

**V1** .. **V2** - диапазон начальной скорости

**T2** - время жизни болванчика

**Х** - радиус, в котором проверяется наличие вражеских ракет при проверке активировать способность «контр-меры» или нет

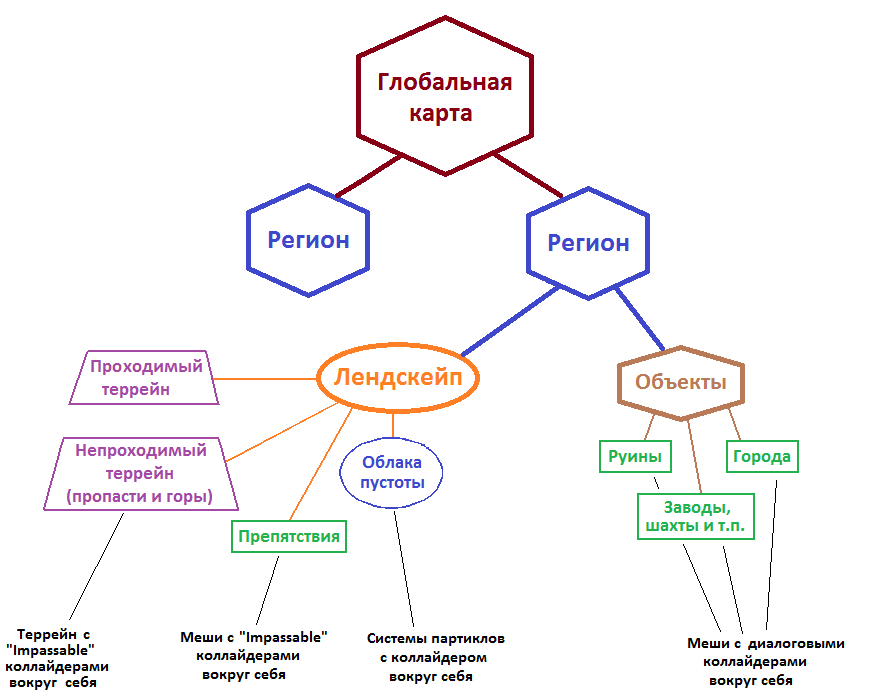
**Y** - радиус от болванчика, залетая в который ракеты проверяют, становится ли этот болванчик их новой целью

**Z** - вероятность того, что болванчик захватит ракету

# Мир

## 9.1. Генерация мира

Начнём с глобал-мапы. Общая её структура такая:



То есть карта – это регионы, состоящие из лендскейпа (террейн и препятствия-меши) и из объектов на нём (меши).

Движение реализуется через нав-меш, в котором непроходимые зоны (террейн или меш) проделывают дыры.

Некоторые зоны могут быть временно непроходимы – на них вешается коллайдер со свойством “Impassable”.

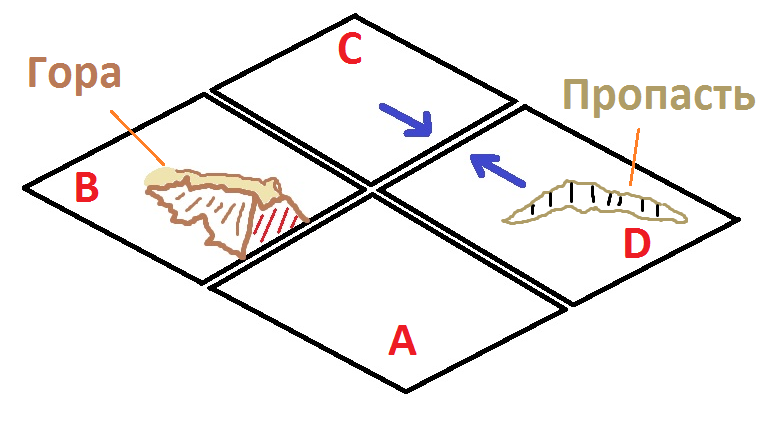
Объекты, с которыми можно взаимодействовать, имеют коллайдер “Contact”.

Теперь как создаётся мир.

Непросто, ведь даже Богу понадобилось на это 6 дней ☺

Итак, последовательность такая:

1. Дизайнер локаций готовит квадраты лендскейпа, которые потом можно сшить в общую карту:



Посередине региона “D” пропасть, но она не доходит до края региона, поэтому не мешает стыку. С куском “B” ситуация другая – гора на нём доходит до края, поэтому его сосед (регион “A”) обязан тоже иметь гору в месте стыка. **///это nice-to-have**

Также стыковаться должны сеттинги.

Таким образом, нам надо ввести следующие свойства сторон регионов:

- сеттинг (пустыня, лес, снег, пограничье)

- тип высоты (обычный, гора типа А, гора типа B и т.д., пропасть типа А, пропасть типа B и т.д.)

- «краевой» (если это свойство “true”, то за этой стороной нет мира и соответственно следующий регион к этой стороне не стыкуется).

Кроме того, самому региону тоже выставляется сеттинг исходя из сеттинга его краёв – и если края имеют разные сеттинги, то и регион имеет все эти сеттинги.

Эти свойства выставляет дизайнер локаций.

1. Из отдельных регионов можно сшивать карту лендскейпа.

Перед началом игры генератор мира этим и занимается:

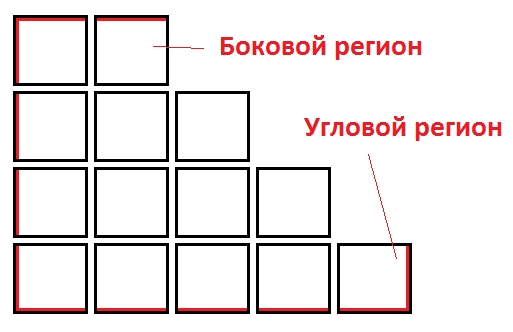
- определяет размер мира (для примера всего 5Х4 регионов)

- ставит угловой регион (у его двух смежных краёв есть свойство «краевой»)

- ставит смежные к нему регионы так, чтоб свойства сторон регионов совпадали

- ставит смежные к этим регионам и так далее

- когда генератор ставит последний в высоту или ширину регион, он выбирает из регионов, имеющих свойство «краевой»:



(легко догадаться, что этот лендскейп строился из левого нижнего угла)

Для начала делаем прямоугольные миры – потом подумаем над круглыми или произвольными.

1. Когда создан лендскейп, генератор расставляет по нему препятствия:

- в предефайненные точки ставятся искусственные препятствия из списка, который прикреплён к типу точки

- препятствия из списка выбираются соответственно сеттингу региона

- они расставляются под рандомными углами в плоскости XOZ  
(возможно стоит попробовать некоторые препятствия наклонять на 5-10 градусов от оси OY)

- объекты со свойством “unrepeatable” не могут находиться ближе, чем указанная у них величина, к такому же объекту   
(между двумя «заводами с красной трубой» со свойством “unrepeateable=100” не может быть меньше чем 100 единиц дистанции)

- естественные (природные) препятствия ставятся на рандомные точки террейна (а не в предефайненные точки), под рандомными углами

- расстояния между естественными препятствиями берутся рандомные, однако вероятность, что 2 одинаковых возникнут рядом должна быть минимальна

Непроходимый террейн даст возможность делать карту высот в некоторых регионах, а препятствия в виде мешей позволят сделать случайно генерируемый лабиринт.

**Конструктор города**

У каждой расы есть своё правило распределения точек **Foundation**

## 9.2. Репутация игрока

## 9.3. Активность фракций

## 9.4. …