**Создание новой игры:**

* После накопления 8 игроков очередь автоматически создает новую комнату с интерфейсом RoyalRoom и передает ей свойства объекта виде QueueGamer (в базовом виде ссылка на ClientServer ник и пароль для возвращения в аккаунт)
* Передает созданную комнату в хранилище активных комнат Rooms
* Запускает комнату вызовом метода StartRoom

**Конструктор IRoom**

* Создает объект IRoomLogic
* Передаем объект IRoomLogic в качестве параметра конструктора INetwork

Класс RoomNetwork – управляет отправкой данных клиенту, содержит в себе список клиентов

**Движение игрока:**

* Начинается с получения сообщения GoToMsg (содержит в себе направление движения)
* При получении GoToMsg оно трансформируется в StartMoveGamer содержащее направление движения и добавляется в очередь сообщений объекта игрока
* После истечения кванта времени оно передается компоненту Movement
* Обработчик этого сообщения в Movement добавляет направление переданное в StartMoveGamer в currentDirection, компоненте переходим в состоянии active
* После перехода в активное состоянии, компонент на каждое сообщение TimeQuantPassed передвигает игрока на некоторое расстояние согласно текущему выбранному направлению.
* При получении сообщения EndMoveGamer компонент Movement переводит active в false(движение игрока прекращается)

**Обработка столкновений с предметами в процесс движения:**

* Получаем сообщение CollisionObjects
* Отодвигаем игрока на необходимое расстояние, чтобы не нарушать границы другого объекта

Каждый 2 ход (10 раз в секунду) в очередь сообщения об произошедших событиях в объекте RoyalGameModel должно добавляться сообщение об изменении координат игрока (сообщение содержит ссылку на объект игрока и точку в которую переместился игрок), которое затем конвертируется (при его обработке в NetworkRoom) в сообщение об изменении координат игрока (содержит координаты и идентификатор игрока (уникален только в рамках игровой комнаты)) – рассылается всем игрокам, попадающее в область видимости этого события

**Механизм работы срочных сообщений**

* В модели хранится коллекция HappenedEvents, в которую добавляются объекты событий, произошедших в игре. Объекты событий создаются в компонентах игровых объектов.
* Так как коллекция ObservalableCollection, то на нее можно подписаться, чтобы узнать о том, что она изменилась. На нее подписывается метод класса RoomNetwork
* При получении такого сообщения INetwork определяет, игроков, которые оказались в границах видимости данного события и уведомляет их об этом событии (конвертируя в нужный тип IMessage)

**Инвентарь. Поднятие предметов**

* Представлен компонентом Collector
* Содержит в себе две коллекции Модификаторы и Оружие
* При получении сообщения PickUpLootMsg мы обращаемся к клеткам карты на которых находится объект (для этого нам нужна ссылка на SolidBody этого объекта) – идем по списку объектов на всех клетках, и определяем:
  + - Является ли данный объект поднимаемым
    - Входит ли игрок в область поднятия игрового предмета
* Мы добавляем его в инвентарь и удаляем с карты. Отправляем этому объекту сообщение о том, что он подобран (обрабатывать его не обязательно)

Реализация проверки на возможность подбора игроком осуществляется с помощью метода CheckCollision

Кроме 2 коллекций инвентарь имеет ссылку на текущие выбранное оружие. При поднятии оружие оно по умолчанию становится выбранным. Чтобы выбрать другое оружие клиент посылает сообщение ChoiceAnotherWeapon(хранит номер нового выбранного оружия). После получения этого события оно конвертируется в ChoiceWeapon (хранит также номер оружия). После получения этого события и проверки его на корректность инвентарь меняет ссылку на выбранное оружие

**Стрельба**

**Оружие**

Содержит следующие компоненты:

* SolidBody – прозрачный
* Shot для создания выстрела(добавляет пулю в коллекцию игровых объектов) содержит в себе компонент Magazin
* Magazin – выполняет все действия происходящие с магазином

**Работа компонента Magazin:**

* Хранит в себе List заданного размера (аргумент конструктора) для хранения магазина
* Метод GetBullet возвращает объект пули заданного типа (тип пули передается в аргументах) и запускает таймер на перезарядку оружия (длительность таймера задается в аргументах конструктора). Если таймер еще не истек данный метод возвращает null и, следовательно выстрел не может быть произведен
* Объект обрабатывает событие MakeReload – перезарядить магазин. Длительность перезарядки передается как аргумент конструктора.
* Если в магазине нет патронов, то производится автоматическая перезарядка
* Магазин должен создавать события начала перезарядки и конца перезарядки магазина (данные о перезарядке отправляются только самому игроку)
* При перезарядке создаются патроны заданного типа(тип передается в аргументах конструктора)

**Работа компонента Shot (пули имеют бесконечную скорость и не имеют тела)**

* Компонент хранит ссылку на компонент Magazin
* Выстрел производится при получении сообщения MakeShot(хранит угол поворота снаряда)
* Компонент запрашивает у Magazin патрон и если он его получает, то начинается поиск игроков стоящих на пути полета пули:

**Алгоритм поиска объектов на пути пули:**

* Получаем уравнение прямой (является объектом) по которой движется пуля [(ссылка на уравнение)](https://zaochnik.com/spravochnik/matematika/prjamaja-ploskost/uravnenie-prjamoj-s-uglovym-koeffitsientom/) – (y – y0) = k \*(x –x0)
* Определяем клетки через которые проходит прямая по формуле Ð¤Ð¾ÑÐ¼ÑÐ»Ð° Ð´Ð»Ð¸Ð½Ñ Ð¾ÑÑÐµÐ·ÐºÐ°. - берем границу следующей клетки и если дальность полета пули больше, то добавляем клетку в массив клеток
* Идем по списку объектов на клетке и ищем те, которым можно нанести урон(разрушаемые - перечисление)
* Если да, то вызываем перегрузку функции CheckCollision – принимающее в качестве аргумента объект уравнения прямой – если объект находится на линии пули, то ему отправляется сообщение GotDamage (хранит количества урона)
* В коллекцию событий отправляется сообщение о том, что произошел выстрел
* Так как пуля не может поразить больше 1 игрока, то проверка на столкновении рассчитывается последовательно(от ближней клетки к дальней): отбираются все объекты с которыми может произойти столкновение, затем ищется самый ближний объект и ему наносится урон и на этом расчеты прекращаются

**Пули**

* Реализуют интерфейс IBullet – имеет свойства: дальность полета пули, урон наносимый пулей
* Являются полностью неизменяемыми

**Работа компонент Healthy – отвечает за здоровье у игровых объектов:**

* Хранит в себе количества HP у данного объекта (передается в качестве аргумента конструктора)
* При получении сообщения GotDamage уменьшает количество HP, если их меньше 0 отправляем сообщение HappenedDeath объекту родителю

**Обработка смерти игрового объекта:**

**Реализация создания игровой комнаты:**

* Справочная информация по работе комнаты: <https://habr.com/ru/post/147431/>
* Создать объект RoyalRoom и передать ему список подключившихся игроков
* Удалить ссылки на игроков в очереди
* Добавить объект в Rooms

**При создании RoyalRoom должны выполняться следующие действия:**

* Создание объекта RoyalRoomLogic
* Создание RoomNetwork

**При создании RoyalRoomLogic должны выполняться следующие действия:**

* Создание контекста данных RoyalGameModel
* Создание таймера запускающегося каждые 50 милисекунд и вызывающего метод TickQuantTimer
* Cоздание класса для хранения времени прошедшего с предыдущего запуска TickQuantTimer
* Реализовать метод TickQuantTimer

**Создание контекста данных RoyalGameModel:**

* Создаем массив клеток размером lengthOfSide. В каждую клетку помещаем ссылку на CellField
* Запускаем метод CreateGameObject (необходимо реализовать по умолчанию) – создаем объекты и добавляет их в список объектов и на игровую карту в соответствии с координатами
* Создаем метод CreatePlayers() – добавляет игроков в спиcки Players, GameObjects и на игровую карту

**При создании RoomNetwork должны выполняться следующие действия**:

* Сохранение ссылки на игровую логику
* Запуск метода CreateClients – создает INetworkClient и записывает в него ссылку на игрока из RoyalRoomLogic
* Создание таймер, который срабатывает каждые 3 сек и вызывает метод HandlerTotalSinch

**Работа механизма столкновений физических тел:**

* При движении игрока или другого физического предмета(пули), вызывается метод AppendCoords компонента SolidBody, в данном методе:
  + - Меняются координаты SolidBody
    - Меняются координаты объекта на карте
    - Запускается метод определения возможности столкновения: Определяются все тела в области данного объекта. Если на области тела находят другие тела, то происходит столкновение. В случае столкновения тело сдвигается в противоположном перемещению направлении, на расстояние достаточное для выхода из области наложения тел предметов.
    - Все столкнувшимся объектам рассылаются сообщения об этом

На карте располагаются не сами объекты, а их компоненты SolidBody, которые наследуют интерфес IFieldObject