OOA и OOD

На практическом примере

Постановка задачи

- В системе моделируется противоборство двух колоний. Колония состоит из строений и юнитов (живых существ).
  - Юнит обладает набором характеристик: на сколько клеток игрового поля он «видит», уровень «здоровья», сколько «здоровья» он теряет во время боя за один удар и др.
  - Юниты разделяются на два типа: строящие и воюющие. Строители наносят врагу незначительный урон и обладают низкой жизнестойкостью. Воины наносят более существенный урон и более защищены. Урон, наносимый воином, меняется случайно в незначительных пределах. Юнит обладает характерным поведением: строитель – если видит врага, – убегает, воин вступает в бой и т.д.
  - У строений есть свои характеристики: размер, количество ударов, после которых строение рухнет и т.д. Строения могут быть двух видов: дома юнитов и защитные укрепления. Каждый дом обеспечивает жизнедеятельность N юнитов. Если вместимости домов не хватает, новый юнит не может быть порожден. Если дом разрушен противоборствующей стороной, то юниты пострадавшей стороны начинают голодать.
  - Дома можно строить без ограничений, на строительство дома нужно достаточно много времени. Юнит создается быстрее, количество ограничено вместительностью строений.
  - Игра происходит на игровом поле. Юнит занимает одно место, здание – несколько. Юнит может перейти на соседнее место, если оно не занято зданием или другим юнитом. Игра может проходить в двух режимах: из некоторого начального положения без участия человека (на начало игры уже созданы все здания и юниты) и с участием человека (он управляет юнитами).

Что мы ищем?

- **Сущности** – *кандидаты на моделирование классом*
  - Объект реального мира, существующий независимо от других
  - Имеет отдельные экземпляры, отличающиеся значениями атрибутов (свойств)
- **Свойства** (или атрибуты) – *кандидаты на моделирование полями*
  - Некое свойство сущности
  - Имеет определенную область допустимых значений
- **Связи**
  - Взаимодействия (процессы)
  - Структурные отношения между сущностями

Постановка задачи – ищем сущности

- В системе моделируется противоборство двух колоний. Колония состоит из строений и юнитов (живых существ).
  - Юнит обладает набором характеристик: на сколько клеток игрового поля он «видит», уровень «здоровья», сколько «здоровья» он теряет во время боя за один удар и др.
  - Юниты разделяются на два типа: строящие и воюющие. Строители наносят врагу незначительный урон и обладают низкой жизнестойкостью. Воины наносят более существенный урон и более защищены. Урон, наносимый воином, меняется случайно в незначительных пределах. Юнит обладает характерным поведением: строитель – если видит врага, – убегает, воин вступает в бой и т.д.
  - У строений есть свои характеристики: размер, количество ударов, после которых строение рухнет и т.д. Строения могут быть двух видов: дома юнитов и защитные укрепления. Каждый дом обеспечивает жизнедеятельность N юнитов. Если вместимости домов не хватает, новый юнит не может быть порожден. Если дом разрушен противоборствующей стороной, то юниты пострадавшей стороны начинают голодать.
  - Дома можно строить без ограничений, на строительство дома нужно достаточно много времени. Юнит создается быстрее, количество ограничено вместительностью строений.
  - Игра происходит на игровом поле. Юнит занимает одно место, здание – несколько. Юнит может перейти на соседнее место, если оно не занято зданием или другим юнитом. Игра может проходить в двух режимах: из некоторого начального положения без участия человека (на начало игры уже созданы все здания и юниты) и с участием человека (он управляет юнитами).

Кандидаты в сущности

- В системе моделируется противоборство двух **колоний**. **Колония** состоит из **строений** и **юнитов** (живых существ).
  - **Юнит** обладает **набором характеристик**: на сколько **клеток** игрового **поля** он «видит», **уровень** «**здоровья**», сколько «**здоровья**» он теряет во время **боя** за один **удар** и др.
  - **Юниты** разделяются на два **типа**: строящие и воюющие. **Строители** наносят **врагу** незначительный **урон** и обладают низкой **жизнестойкостью**. **Воины** наносят более существенный **урон** и более защищены. **Урон**, наносимый **воином**, меняется случайно в незначительных **пределах**. **Юнит** обладает характерным **поведением**: **строитель** – если видит **врага**, – убегает, **воин** вступает в **бой** и т.д.

Кандидаты в сущности

- У **строений** есть свои **характеристики**: **размер**, количество **ударов**, после которых **строение** рухнет и т.д. **Строения** могут быть двух **видов**: **дома юнитов** и защитные **укрепления**. Каждый **дом** обеспечивает **жизнедеятельность** N **юнитов**. Если **вместимости домов** не хватает, новый **юнит** не может быть порожден. Если **дом** разрушен противоборствующей **стороной**, то **юниты** пострадавшей **стороны** начинают голодать.
- **Дома** можно строить без **ограничений**, на **строительство дома** нужно достаточно много **времени**. **Юнит** создается быстрее, **количество** ограничено **вместительностью строений**.
- **Игра** происходит на игровом **поле**. **Юнит** занимает одно **место**, **здание** – несколько. **Юнит** может перейти на соседнее **место**, если оно не занято **зданием** или другим **юнитом**. **Игра** может проходить в двух **режимах**: из некоторого начального **положения** без участия **человека** (на **начало** игры уже созданы все **здания** и **юниты**) и с **участием человека** (он управляет **юнитами**).

Анализ кандидатов

- **Колония == Сторона** - сущность
- **Строение == Здание** - сущность
- **Юнит** - сущность
- **Набор характеристик** - свойства сущности (общее)
- **Клетка == Место** - сущность
- **(Игровое) Поле** - сущность
- **Уровень** «**здоровья**» - свойство сущности (юнит)
- **Бой** - процесс
- **Удар** - процесс
- **Тип (юнита)** - связь между сущностями
- **Строитель** - сущность
- **Враг** - свойство (принадлежность колонии/стороне) сущности
- **Урон** - свойство сущности
- **Жизнестойкость** - свойство сущности
- **Воин** - сущность
- **Предел (изменения урона)** - свойство свойства сущности
- **Поведение**- свойство сущности

Анализ кандидатов

- **Размер** - свойство сущности
- **Вид (строения)** – связь между сущностями
- **Дом** - сущность
- **Укрепление** - сущность
- **Жизнедеятельность** - процесс
- **Вместимость (домов)** - свойство сущности (Колонии)
- **Ограничение** - процесс (его существенная деталь)
- **Строительство(дома)** - процесс
- **Время** - процесс (его существенная деталь)
- **Количество (юнитов)** - свойство сущности (Колонии)
- **Игра** – сущность (глобальная?)
- **Режим** - свойство сущности (Игры)
- **Положение** - свойство сущности (Игры)
- **Человек** – сущность
- **Начало (игры)** - процесс
- **Участие (человека)** - процесс

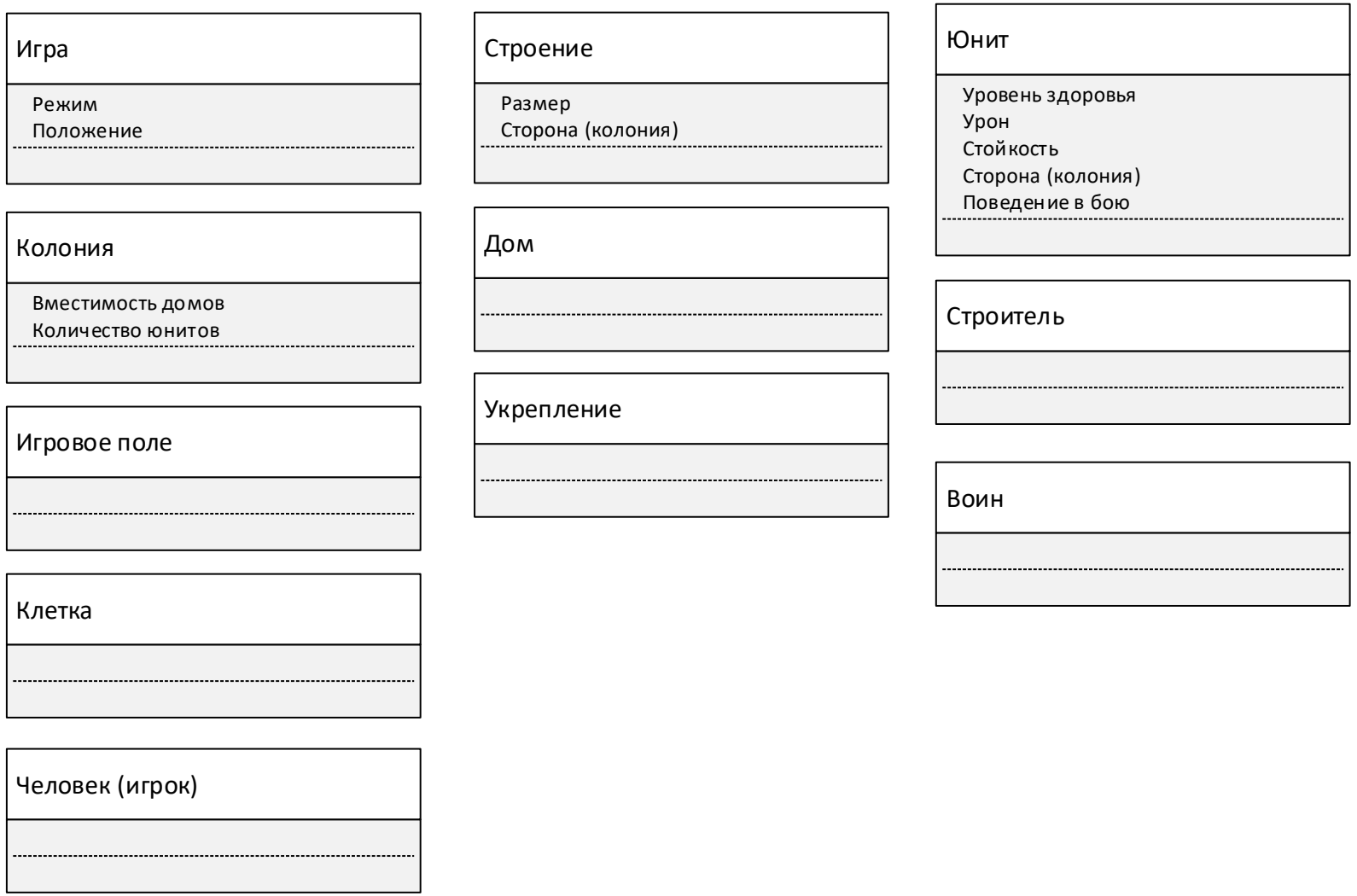
Сущности

- Игра
- Колония
- Игровое поле
- Клетка
- Юнит
- Строитель
- Воин
- Строение
- Дом
- Укрепление
- Человек (Игрок)

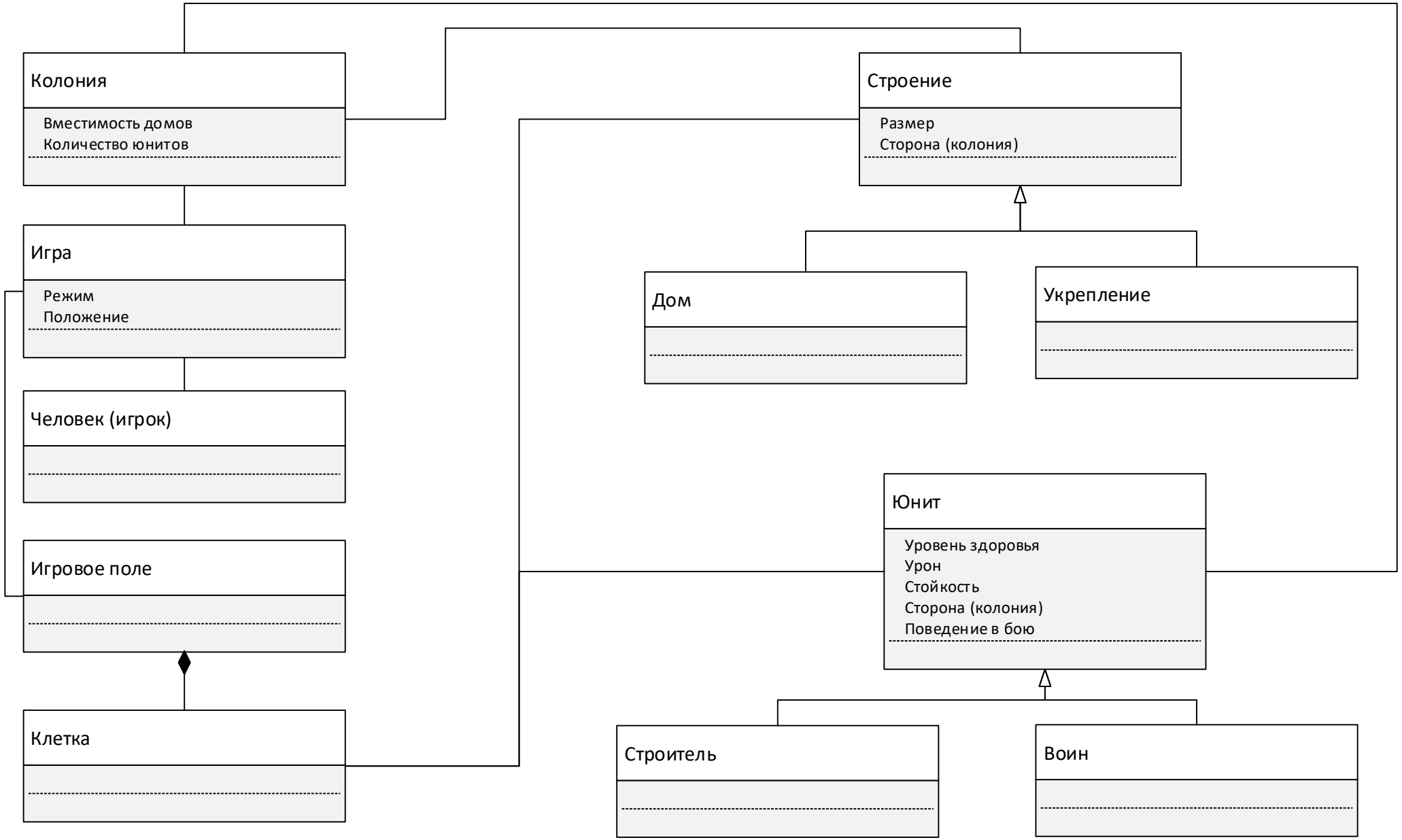
Сущности и свойства/действия

- Игра
  - Режим
  - Положение
- Колония
  - Вместимость домов
  - Количество юнитов
- Игровое поле
- Клетка
- Строение
  - Размер
  - Сторона (колония)
- Дом
- Укрепление
- Человек (Игрок)
- Юнит
  - Уровень здоровья
  - Урон
  - Стойкость
  - Сторона
  - Поведение в бою
- Строитель
- Воин

Диаграмма классов для сущностей



Связи между сущностями



Постановка задачи – ищем свойства

- В системе моделируется противоборство двух колоний. Колония состоит из строений и юнитов (живых существ).
  - Юнит обладает набором характеристик: на сколько клеток игрового поля он «видит», уровень «здоровья», сколько «здоровья» он теряет во время боя за один удар и др.
  - Юниты разделяются на два типа: строящие и воюющие. Строители наносят врагу незначительный урон и обладают низкой жизнестойкостью. Воины наносят более существенный урон и более защищены. Урон, наносимый воином, меняется случайно в незначительных пределах. Юнит обладает характерным поведением: строитель – если видит врага, – убегает, воин вступает в бой и т.д.
  - У строений есть свои характеристики: размер, количество ударов, после которых строение рухнет и т.д. Строения могут быть двух видов: дома юнитов и защитные укрепления. Каждый дом обеспечивает жизнедеятельность N юнитов. Если вместимости домов не хватает, новый юнит не может быть порожден. Если дом разрушен противоборствующей стороной, то юниты пострадавшей стороны начинают голодать.
  - Дома можно строить без ограничений, на строительство дома нужно достаточно много времени. Юнит создается быстрее, количество ограничено вместительностью строений.
  - Игра происходит на игровом поле. Юнит занимает одно место, здание – несколько. Юнит может перейти на соседнее место, если оно не занято зданием или другим юнитом. Игра может проходить в двух режимах: из некоторого начального положения без участия человека (на начало игры уже созданы все здания и юниты) и с участием человека (он управляет юнитами).

Свойства сущностей

- В системе моделируется противоборство двух колоний. Колония состоит из строений и юнитов (живых существ).
  - Юнит обладает набором характеристик: **на сколько клеток игрового поля он «видит»**, **уровень «здоровья»**, **сколько «здоровья» он теряет во время боя за один удар** и др.
  - Юниты разделяются на два типа: строящие и воюющие. Строители наносят врагу **незначительный урон** и обладают **низкой жизнестойкостью**. Воины наносят более **существенный урон** и более **защищены**. Урон, наносимый воином, **меняется случайно в незначительных пределах**. Юнит **обладает характерным поведением**: строитель – если видит врага, – убегает, воин вступает в бой и т.д.

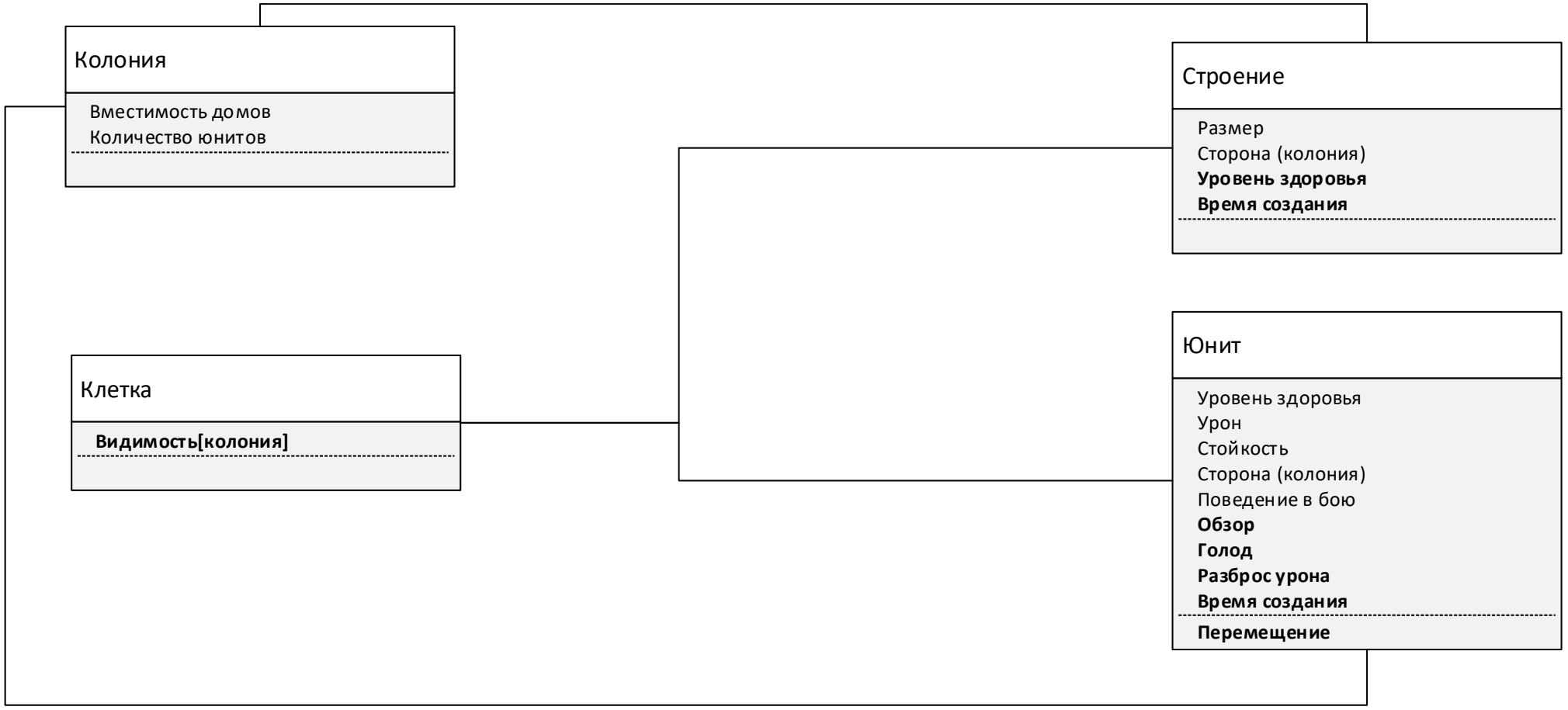
Свойства сущностей

- У строений есть свои характеристики: **размер**, **количество ударов, после которых строение рухнет** и т.д. Строения могут быть двух видов: дома юнитов и защитные укрепления. Каждый дом обеспечивает жизнедеятельность N юнитов. Если вместимости домов не хватает, новый юнит не может быть порожден. Если дом разрушен противоборствующей стороной, то юниты пострадавшей стороны **начинают голодать**.
- Дома можно строить без ограничений, на **строительство дома нужно достаточно много времени**. Юнит создается **быстрее**, количество ограничено **вместительностью строений**.
- Игра происходит на игровом поле. Юнит занимает одно место, здание – несколько. Юнит **может перейти** на соседнее место, если оно не занято зданием или другим юнитом. Игра может проходить в двух режимах: из некоторого начального положения без участия человека (на начало игры уже созданы все здания и юниты) и с участием человека (он управляет юнитами).

Новые свойства сущностей

- Юнит
  - Обзор (отсюда следует неявное требование реализации «тумана войны»)
  - Голод
  - Разброс урона
  - Время создания
  - Перемещение
- Клетка
  - Видимость определенной стороне
- Строение
  - Уровень прочности («здоровье»)
  - Время создания

Внесем изменения на диаграмму



Вводим новую сущность

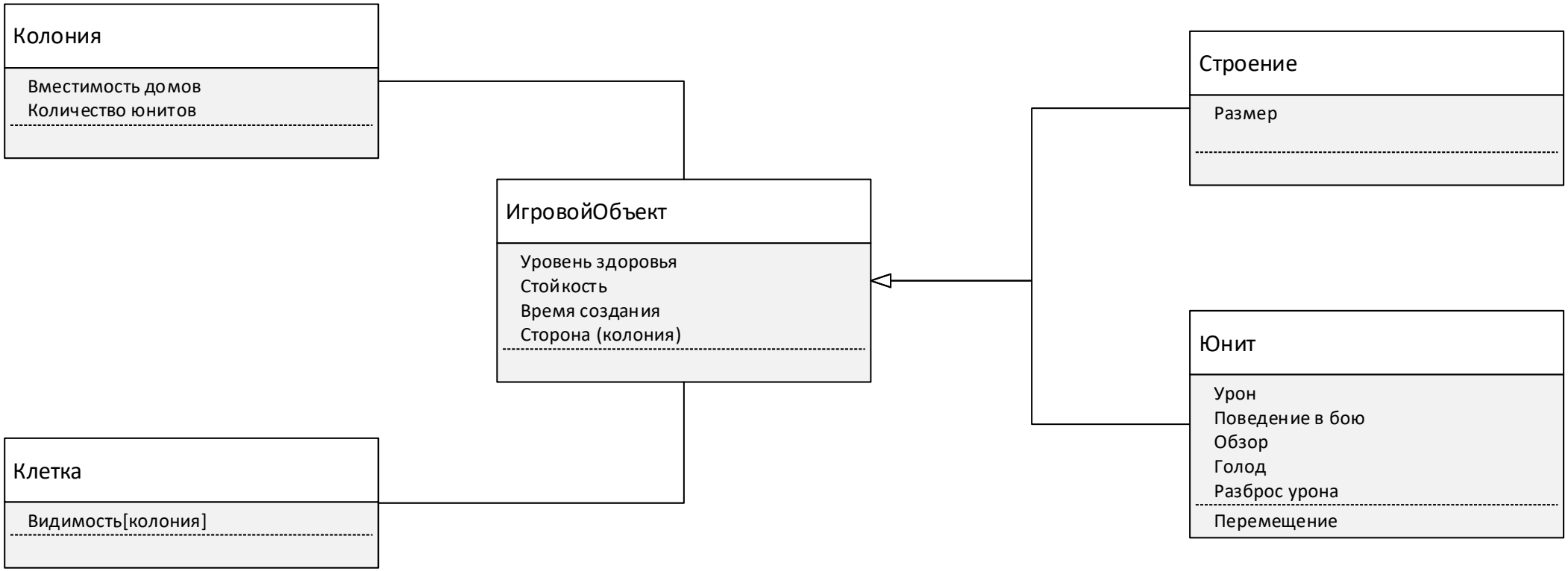
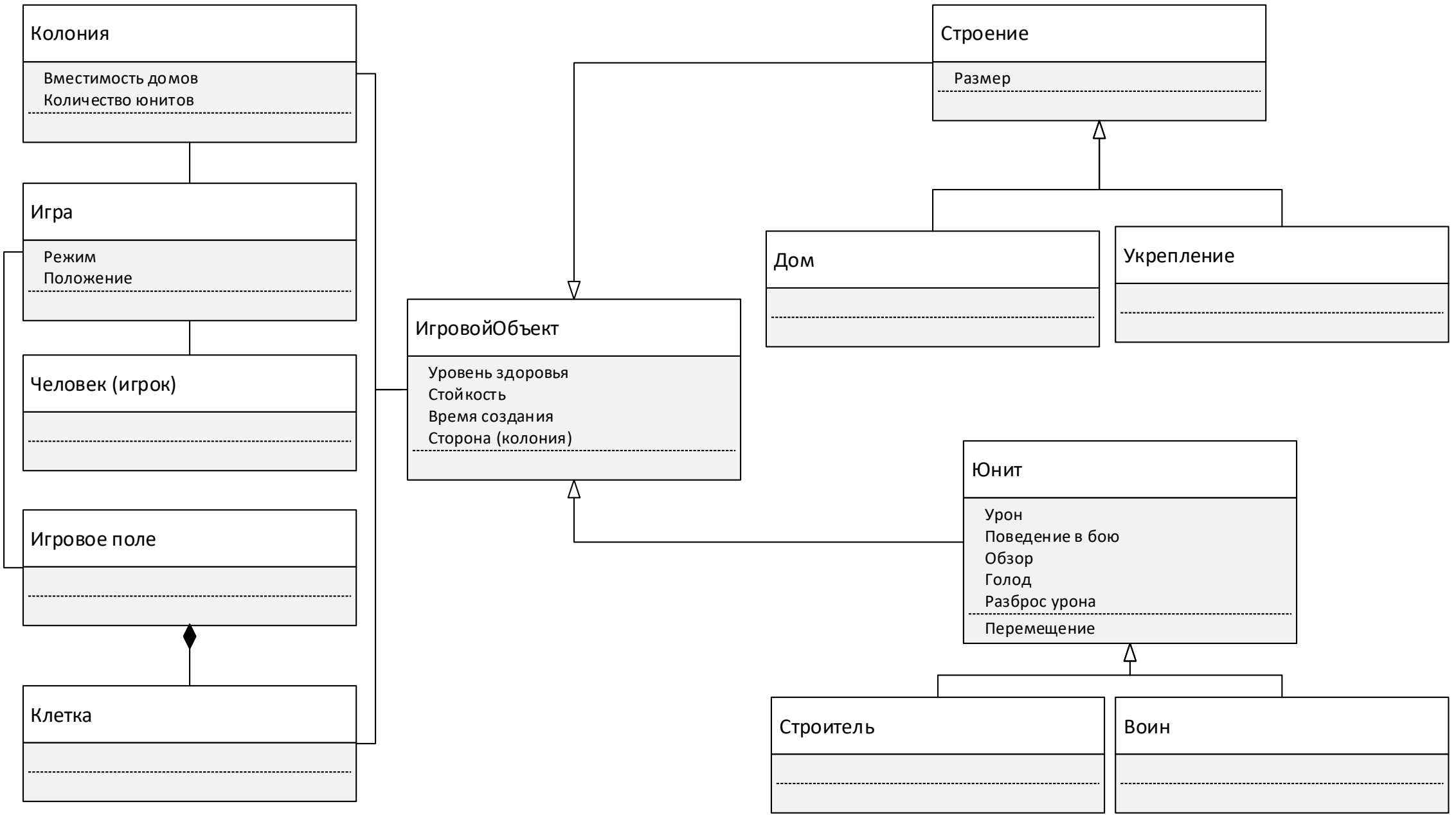


Диаграмма классов предметной области



Постановка задачи – ищем взаимодействия

- В системе моделируется противоборство двух колоний. Колония состоит из строений и юнитов (живых существ).
  - Юнит обладает набором характеристик: на сколько клеток игрового поля он «видит», уровень «здоровья», сколько «здоровья» он теряет во время боя за один удар и др.
  - Юниты разделяются на два типа: строящие и воюющие. Строители наносят врагу незначительный урон и обладают низкой жизнестойкостью. Воины наносят более существенный урон и более защищены. Урон, наносимый воином, меняется случайно в незначительных пределах. Юнит обладает характерным поведением: строитель – если видит врага, – убегает, воин вступает в бой и т.д.
  - У строений есть свои характеристики: размер, количество ударов, после которых строение рухнет и т.д. Строения могут быть двух видов: дома юнитов и защитные укрепления. Каждый дом обеспечивает жизнедеятельность N юнитов. Если вместимости домов не хватает, новый юнит не может быть порожден. Если дом разрушен противоборствующей стороной, то юниты пострадавшей стороны начинают голодать.
  - Дома можно строить без ограничений, на строительство дома нужно достаточно много времени. Юнит создается быстрее, количество ограничено вместительностью строений.
  - Игра происходит на игровом поле. Юнит занимает одно место, здание – несколько. Юнит может перейти на соседнее место, если оно не занято зданием или другим юнитом. Игра может проходить в двух режимах: из некоторого начального положения без участия человека (на начало игры уже созданы все здания и юниты) и с участием человека (он управляет юнитами).

Ищем взаимодействия

- В системе моделируется **противоборство** двух колоний. Колония состоит из строений и юнитов (живых существ).
  - Юнит обладает набором характеристик: на сколько клеток игрового поля он «**видит**», уровень «здоровья», сколько «здоровья» он **теряет** во время боя за один **удар** и др.
  - Юниты разделяются на два типа: **строящие** и **воюющие**. Строители **наносят** врагу незначительный урон и **обладают низкой жизнестойкостью**. Воины **наносят** более существенный урон и **более защищены**. Урон, наносимый воином, меняется случайно в незначительных пределах. Юнит обладает характерным **поведением**: строитель – если видит врага, – **убегает**, воин **вступает в бой** и т.д.

Ищем взаимодействия

- У строений есть свои характеристики: размер, количество ударов, после которых строение **рухнет** и т.д. Строения могут быть двух видов: дома юнитов и защитные укрепления. Каждый дом **обеспечивает** жизнедеятельность N юнитов. Если вместимости домов не хватает, новый юнит не может быть **порожден**. Если дом **разрушен** противоборствующей стороной, то юниты пострадавшей стороны начинают **голодать**.
- Дома можно **строить** без ограничений, на строительство дома нужно достаточно много времени. Юнит **создается** быстрее, количество ограничено вместительностью строений.
- Игра происходит на игровом поле. Юнит занимает одно место, здание – несколько. Юнит может **перейти** на соседнее место, если оно не занято зданием или другим юнитом. Игра может **проходить** в двух режимах: из некоторого начального положения без участия человека (на начало игры уже созданы все здания и юниты) и с участием человека (он **управляет** юнитами).

Процессы

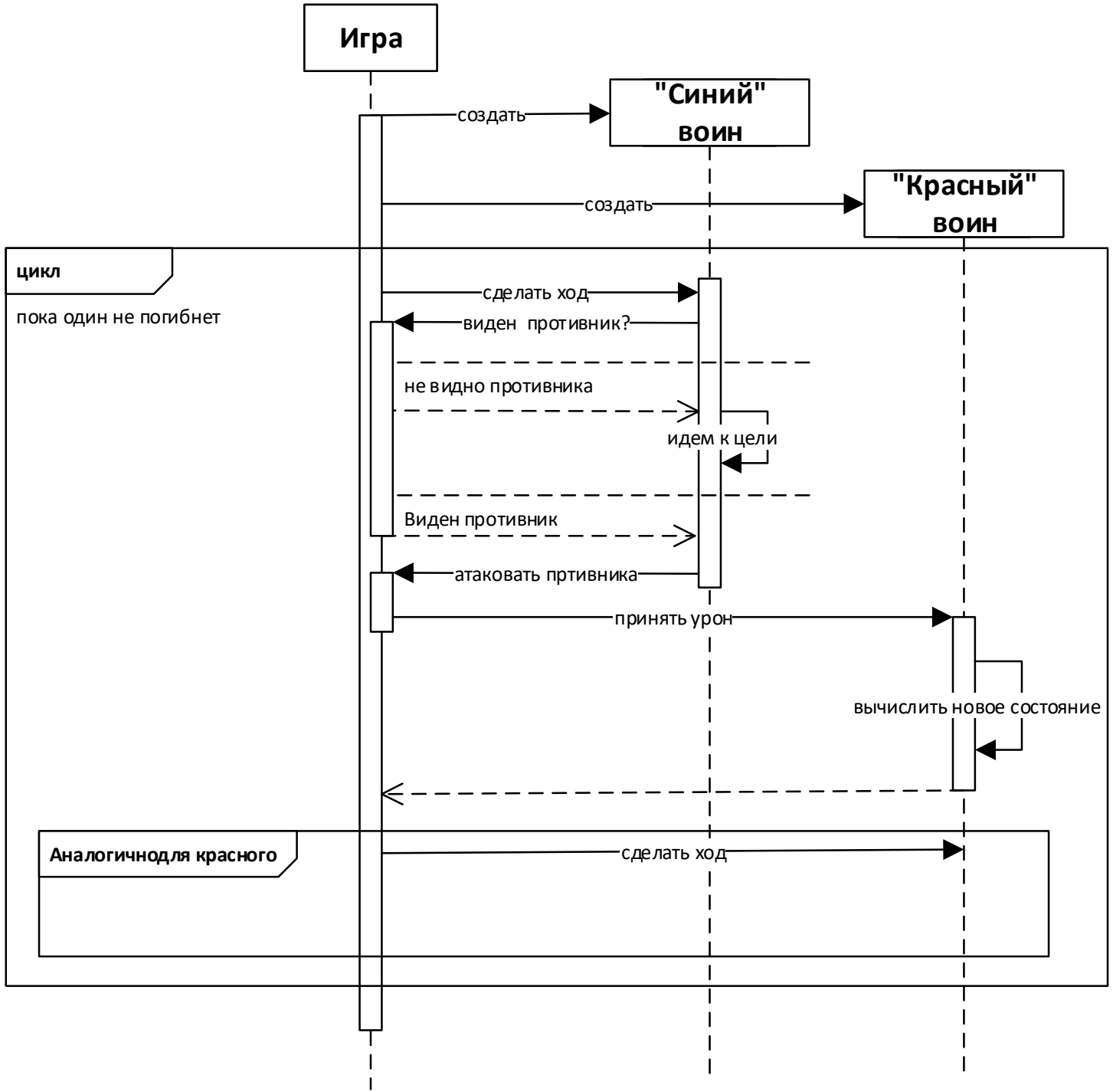
- Игра
  - Противоборство – запуск модели
  - Создать юнит/построить здание
- Юнит
  - Видеть – пометить клетки как видимые (для отрисовки и AI)
  - Терять здоровье – принимать урон. Расчет потерянного здоровья в зависимости от стойкости.
  - Ударять – наносить урон. Отправить сообщение о том, что удар нанесен (*получателю или игровому контроллеру, об этом позже*)
  - Строить – приступить к созданию строения
  - Вступить в бой (атаковать) – приступить к сближению и нанесению урона.
  - Рухнуть/погибнуть – изменить состояние на «мертв», отрисовать смерть объекта, возможно, через какое-то время, удалить объект вообще.
  - Голодать – изменить состояние на «голоден» (влияет на др. расчеты модели)
  - Перейти – сменить текущую клетку.
  - Ожидать команды от человека.

Простейший вариант использования – «дуэль»

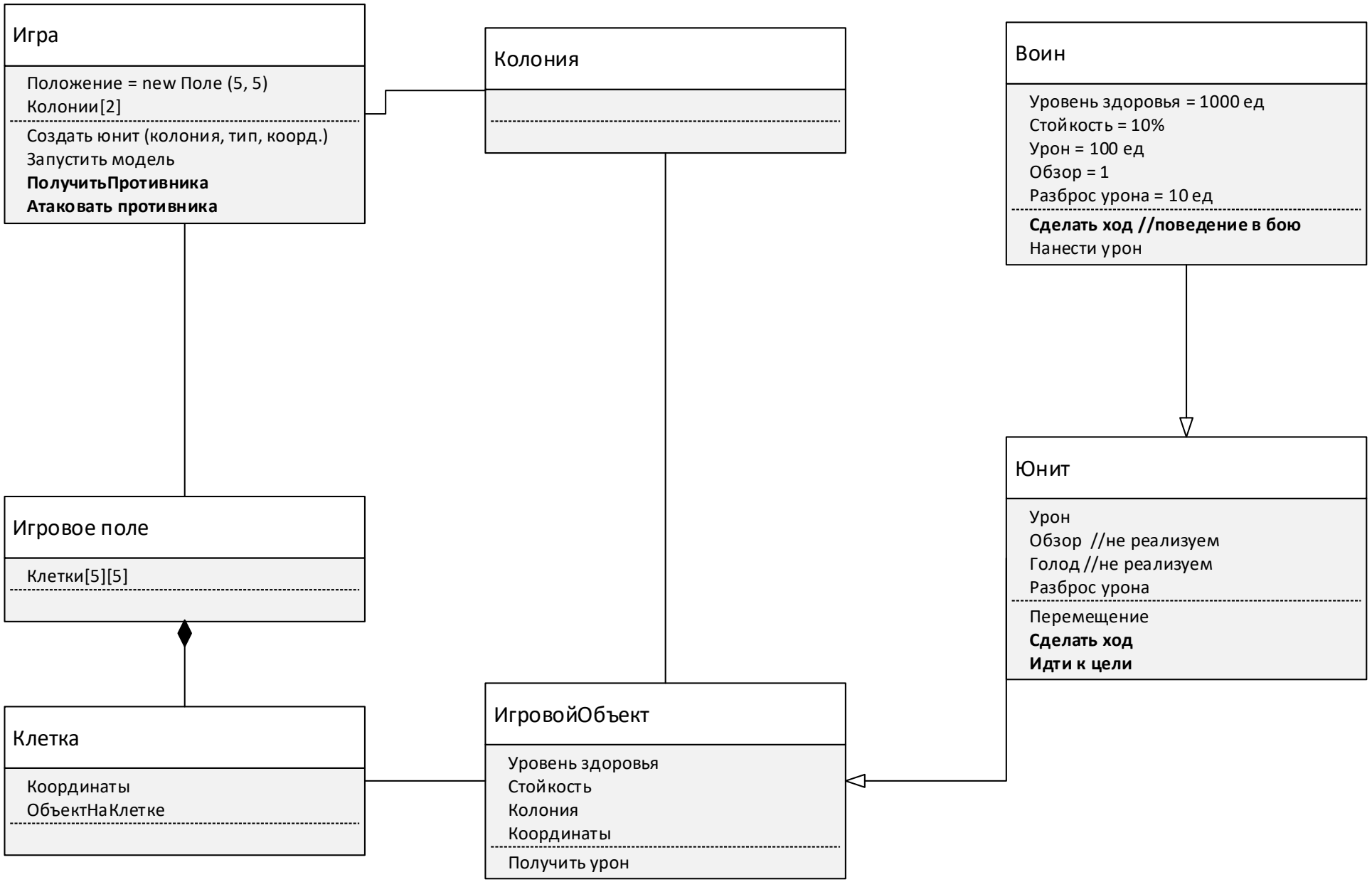
- Начальные условия
  - Карта 5 на 5
  - 2 стороны – колонии «синих» и «красных».
  - Каждая колония создает по одному юниту типа «воин» в противоположных углах карты
  - Видимость каждого юнита – 1 клетка.
  - За один ход юнит может либо переместиться на 1 клетку, либо атаковать.
  - Поведение в бою – поиск, сближение и атака.
  - Видимость не учитываем (видим соседние клетки)
- Работа модели
  - Пошаговая передача хода между сторонами до уничтожения одного из юнитов.

Проектируем реализацию

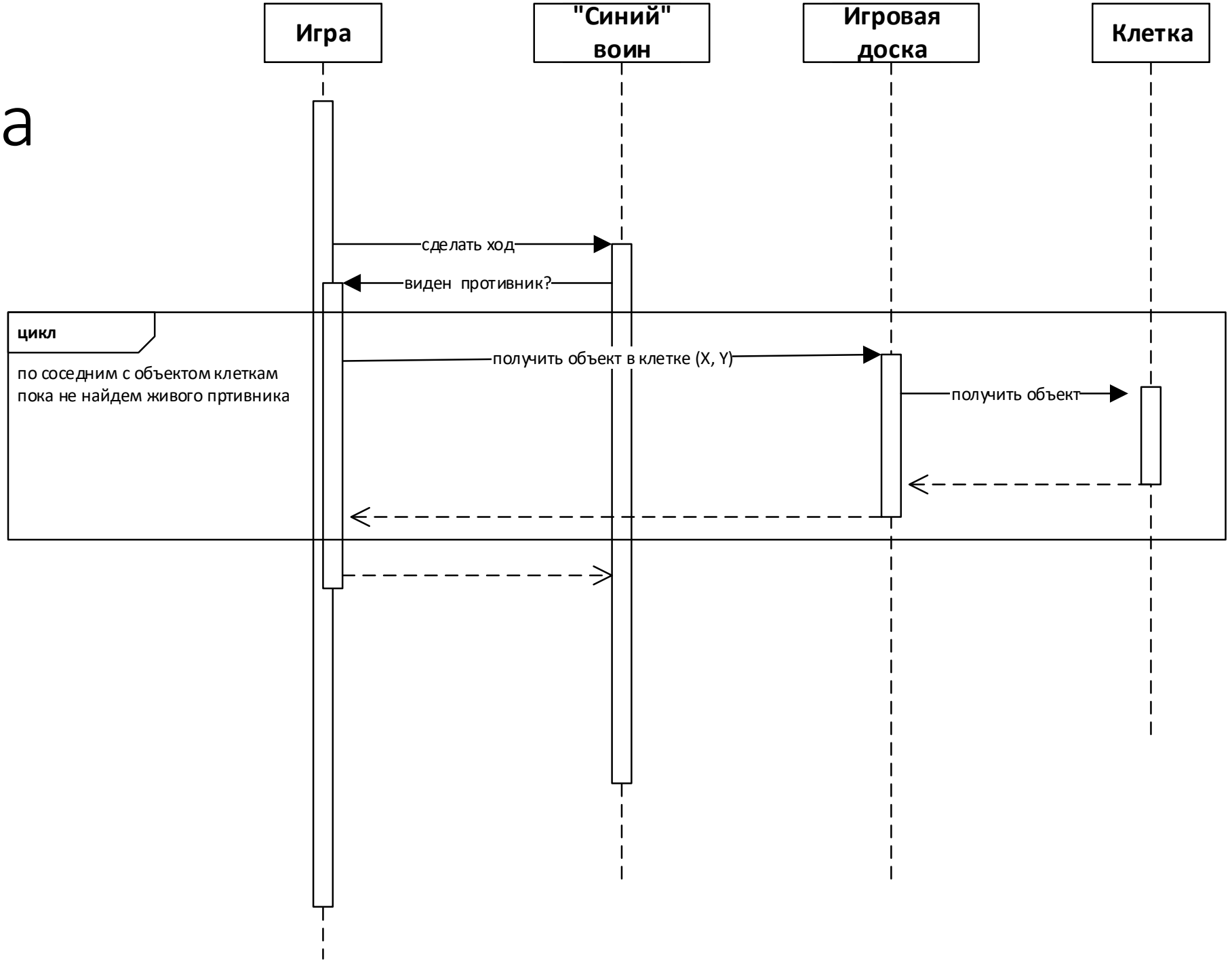
Алгоритм


Игра
"Синий" воин
"Красный" воин
создать
создать
цикл
пока один не погибнет
сделать ход
виден противник?
не видно противника
идем к цели
Виден противник
атаковать пртивника
принять урон
вычислить новое состояние
Аналогичнодля красного
сделать ход

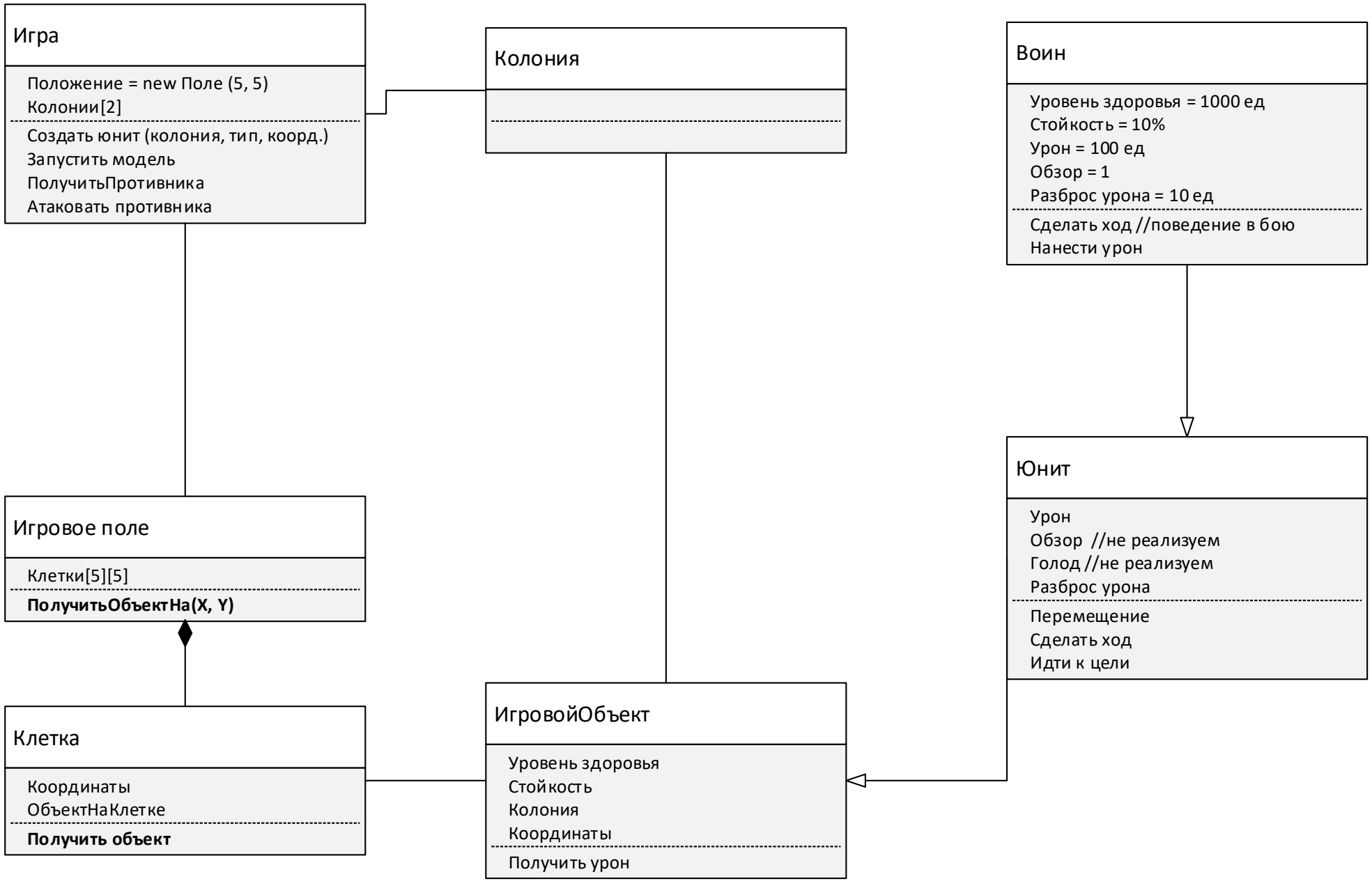
Дополняем классы



Поиск противника

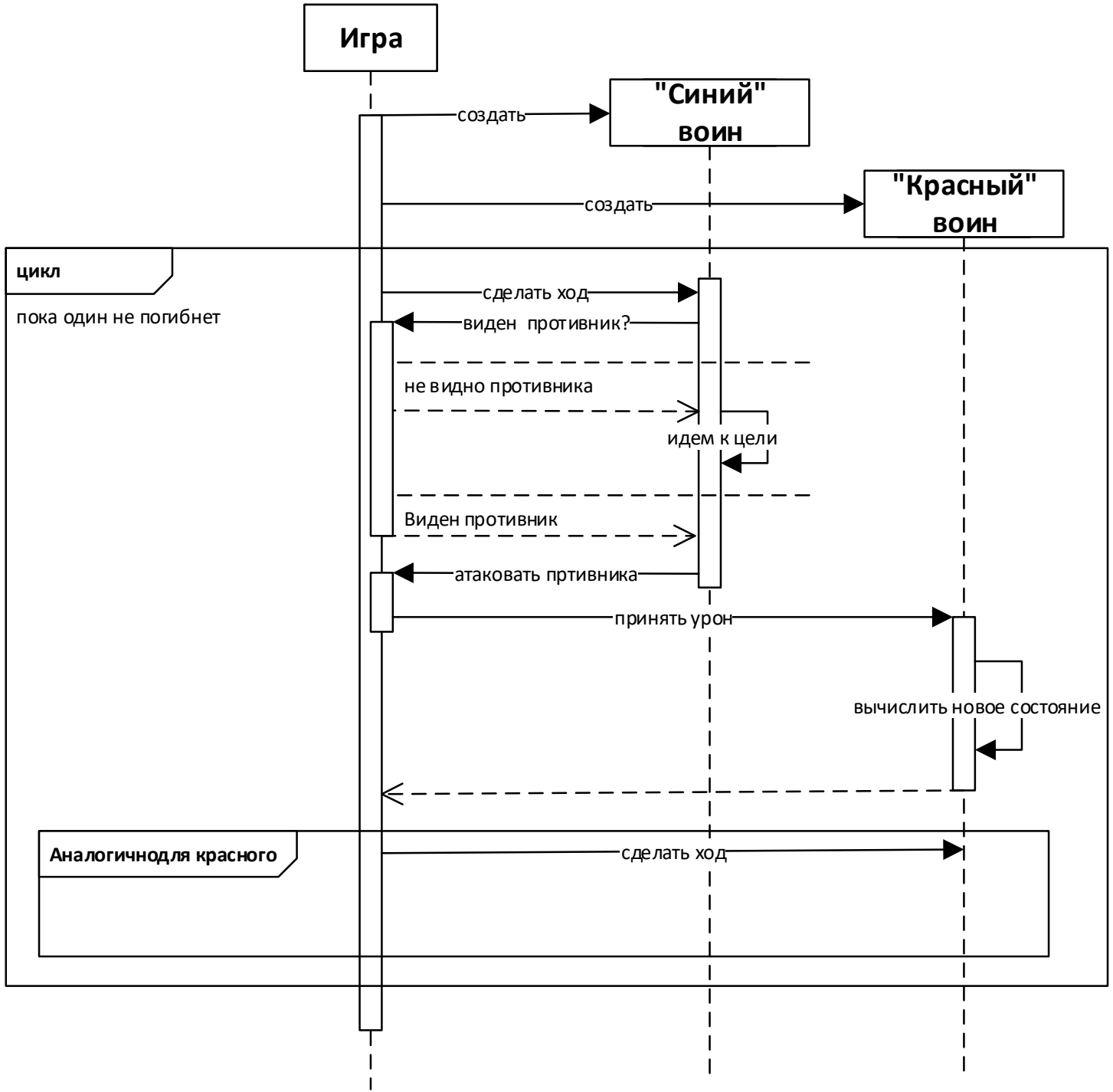


Дополняем классы

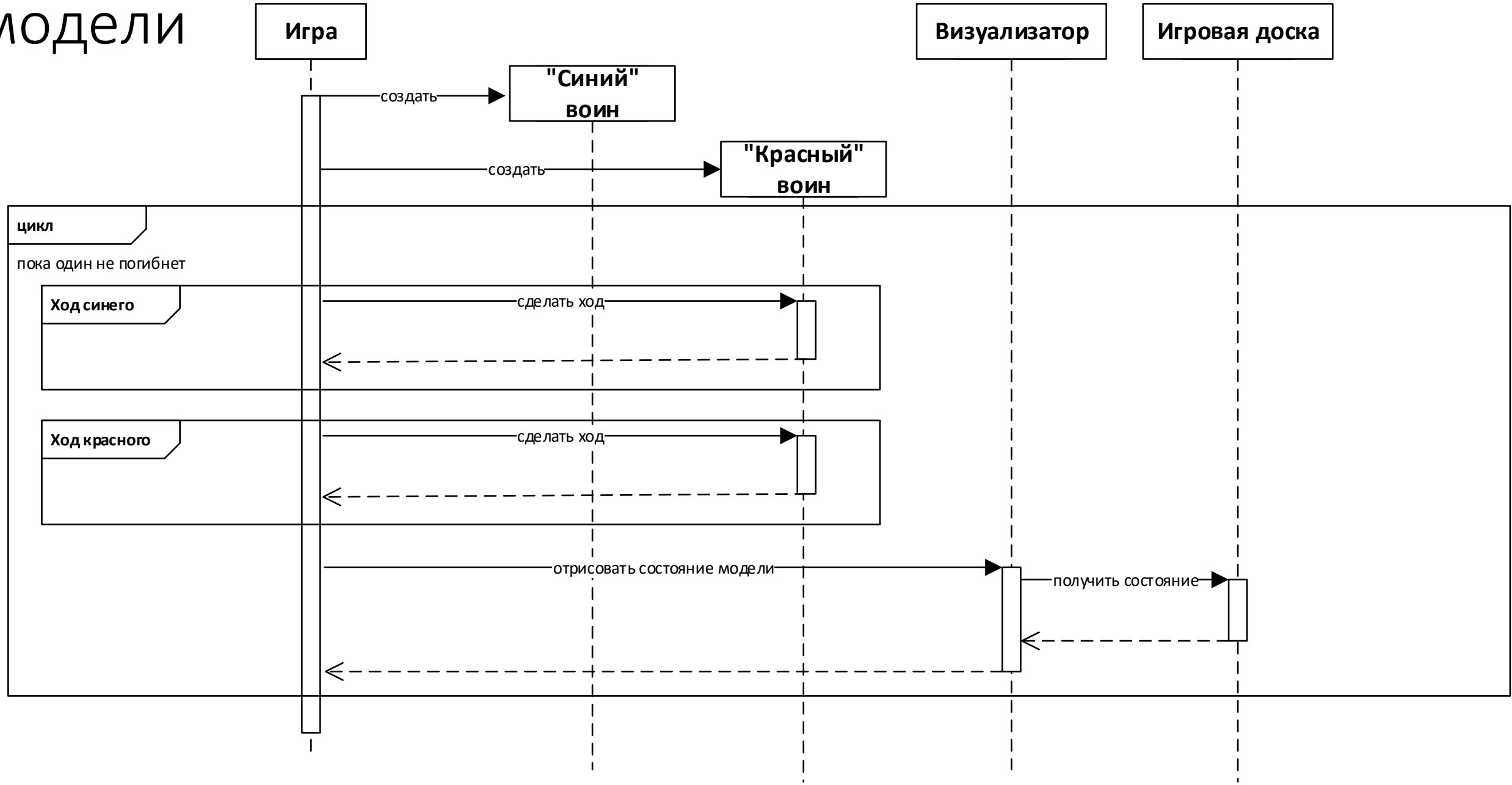


Алгоритм

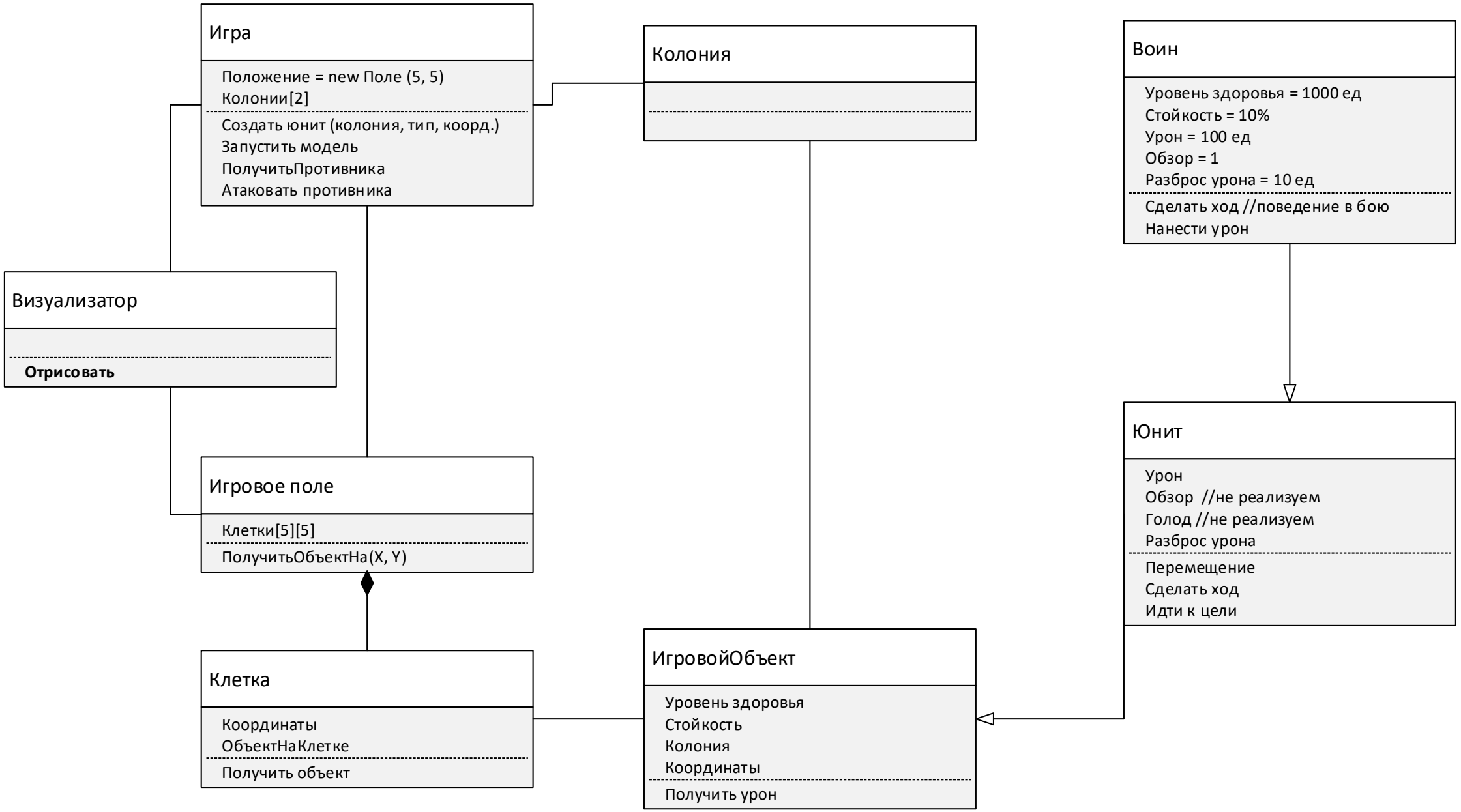
Что еще осталось неохваченным?



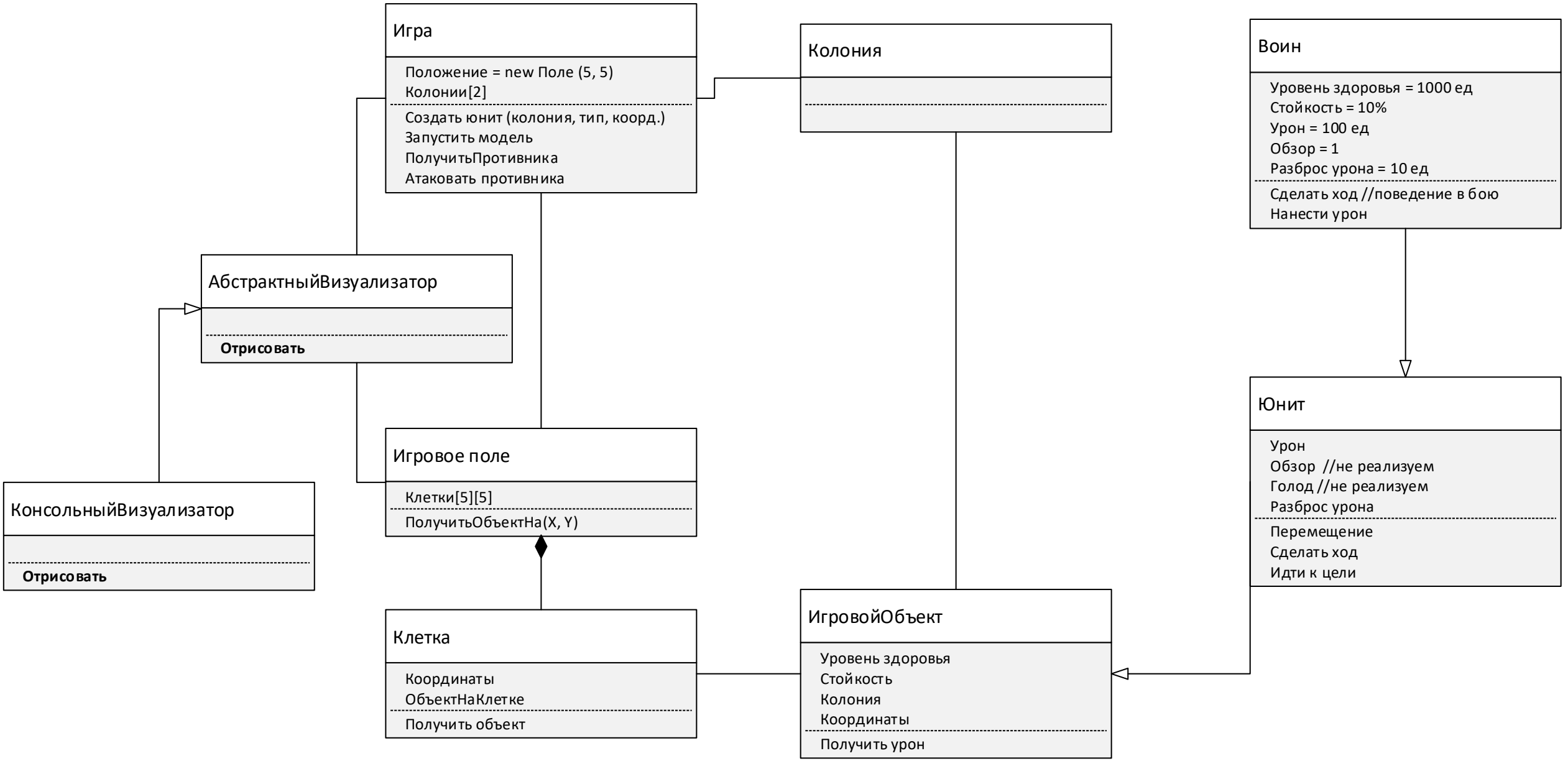
Визуализация модели



Дополняем классы



Дополняем классы



Реализация

- https://github.com/pelipas/OOP_2017/tree/master/2 семестр/OOP_demo/OOP_demo
  - Несмотря на использование C#, стиль кода максимально приближен к C++. Это сделано умышленно, для лучшего понимания большинством обучающихся. Знатоков C# прошу понять и простить ☺
- Ставим Visual Studio Community Edition
- Создаем C# Console Application
- Добавляем классы
  - Game - Игра
  - Board – Игровая доска
  - Square – Клетка игровой доски
  - Colony – Колония (сторона игры)
  - GameObject – Игровой объект
  - Unit - Юнит
  - Warrior - Воин
  - Visualizer – Абстрактный визуализатор
  - ConsoleVisualizer – Консольный визуализатор

Колония

```
class Colony
   {
       private string _name;

       public Colony(string name)
       {
           _name = name;
       }

       public string GetName()
       {
           return _name;
       }
   }
```

Игра

```
class Game
{
        private Board _state;
        private Colony _blueColony;
        private Colony _redColony;

        private Visualizer visualizer = new ConsoleVisualizer();

        //Поведение будет позже, пока только структура
}
```

Игровая доска

```
class Board
{
        private int _sizeX, _sizeY;
        private Square[,] _squares;
        public Board(int sizeX, int sizeY)
        {
            _squares = new Square[sizeX, sizeY];
            for (int i = 0; i < sizeX; i++)
              for (int j = 0; j < sizeY; j++)
                  _squares[i,j] = new Square(i+1, j+1);
            _sizeX = sizeX;
            _sizeY = sizeY;
        }
        public GameObject GetObjAt(int x, int y)
        {
            if(x>=1 && x<=_sizeX && y >= 1 && y<=_sizeY)
                return _squares[x-1, y-1].GetObject();
            return null;
        }
}
```

Клетка

```
class Square
    {
        private int _x, _y;
        private GameObject _obj;

        public Square(int x, int y)
        {
            _x = x;
            _y = y;
            _obj = null;
        }
        public GameObject GetObject() {return _obj; }

        public void SetObject(GameObject obj)
        {
            _obj = obj;
            obj.SetCoords(_x, _y);
        }

        public void RemoveObject() { _obj = null; }
    }
```

Игровой объект

```
class GameObject
{
        protected Game _game;
        protected int _x, _y, Hp, Def;
        protected Colony _side;
        private bool _alive;

        public GameObject(Game game, Colony side)
        {
             _game = game;
             _side = side;
             _alive = true;
             _x = 1; _y = 1;
        }
        public bool IsAlive() { return _alive; }
        public int GetX() { return _x; }
        public int GetY() {return _x; }
        public Colony GetSide() {return _side; }
        public int GetHp() {return Hp; }

//продолжение на следующем слайде
```

Игровой объект

```
//class GameObject - начало на предыдущем слайде

    public void SetCoords(int x, int y)
    {
        _x = x;
        _y = y;
    }

    public void TakeDamage(int dmg)
    {
        if (!_alive) return; //То, что уже умерло, умереть не может
        int resultDmg = dmg - Def; //Поглощаем Def единиц урона
        if (resultDmg > 0) Hp -= resultDmg; //Не допускаем отрицательного урона (лечения)
        if (Hp <= 0) _alive = false; //Если ХП=0, пациент скорее мертв, чем жив
    }
}
```

Юнит

```
class Unit : GameObject
{
    private int _targetX, _targetY;
    protected int AttackPower, AttackDispersion;

    public Unit(Game game, Colony side) : base(game, side) {}

    public void SetTargetLocation(int x, int y)   {_targetX = x; _targetY = y; }

    public virtual void Action() { MoveToTargetLocation(); }

    protected virtual void MoveToTargetLocation()
    {
        int newX = (_targetX == _x) ? _x : (_targetX < _x) ? _x - 1 : _x + 1;
        int newY = (_targetY == _y) ? _y : (_targetY < _y) ? _y - 1 : _y + 1;
        Move(newX, newY);
    }

    public bool Move(int x, int y) { return _game.TryToMove(this, x, y); }

//продолжение на следующем слайде
```

Юнит

```
    //class Unit - начало на предыдущем слайде

    protected virtual void Attack(GameObject enemy)
    {
        //пропускаем доп. проверки пока
        Random rnd = new Random((int)DateTime.Now.Ticks);
        int attackDamage = AttackPower + AttackDispersion*rnd.Next(-100, 100)/100;
        _game.DealDamage(this, enemy, attackDamage);
    }

    protected GameObject FindEnemy()
    {
        return _game.FindNearestEnemy(this);
    }

}
```

Воин

```
class Warrior : Unit
    {
        public override void Action()
        {
            GameObject enemy = FindEnemy();
            if (enemy != null) // «если драка неизбежна – бей первым!» (с)
            {
                Attack(enemy);
            }
            else //иначе продолжаем вести себя как обычный юнит – идем по своим делам
            {
                base.Action();
            }
        }
        public Warrior(Game game, Colony side) : base(game, side)
        {
            this.AttackPower = 100; this.AttackDispersion = 10; this.Def = 50; this.Hp = 200;
        }
    }
```

Функции движка игры (класс Game)

```
public bool TryToMove(GameObject obj, int x, int y)
{
    bool success = false;
    if (Math.Abs(obj.GetX() - x) <= 1 && Math.Abs(obj.GetY() - y) <= 1 ) //на соседнюю
    {
        if (_state.GetObjAt(x, y) == null) //на пустую
        {
            _state.SetObjAt(x, y, obj);
            success = true;
        }
    }
    return success;
 }

public void DealDamage(Unit attacker, GameObject enemy, int attackDamage)
{
    enemy.TakeDamage(attackDamage); //упрощенно, без доп. проверок
}
```

Функции движка игры (класс Game)

```
public GameObject FindNearestEnemy(Unit unit)
{
    for (int i = unit.GetX()-1 ; i <= unit.GetX()+1; i++)
    {
        for (int j = unit.GetY()-1; j <= unit.GetY()+1; j++)
        {
            GameObject candidate = _state.GetObjAt(i, j);
            if (candidate != null && candidate.GetSide() != unit.GetSide() &&
                  candidate.IsAlive())
            {
               return candidate;
            }
        }
    }
    return null;
}
```

Инициализируем модель (класс Game)

```
private List<Unit> _units = new List<Unit>();

public void InitModel()
{
    _state = new Board(BOARD_SIZE_X, BOARD_SIZE_Y);
    _blueColony = new Colony("Blue");
    _redColony = new Colony("Red");

    Warrior blue = new Warrior(this, _blueColony);
    blue.SetTargetLocation(BOARD_SIZE_X/2, BOARD_SIZE_Y/2); //к центру
    _state.SetObjAt(1, 1, blue);

    Warrior red = new Warrior(this, _redColony);
    red.SetTargetLocation(BOARD_SIZE_X / 2, BOARD_SIZE_Y / 2); //к центру
    _state.SetObjAt(BOARD_SIZE_X, BOARD_SIZE_Y, red);

    _units.Add(blue);
    _units.Add(red);
}
```

Запускаем модель (класс Game)

```
private Visualizer visualizer = new ConsoleVisualizer();

public void RunModel()
{
    do
    {
        foreach (Unit unit in _units)
        {
            if (!unit.IsAlive()) return; //бой до первой смерти
            unit.Action(); //делаем ход
            visualizer.Draw(_state); //визуализируем новое состояние после хода
            Thread.Sleep(STEP_DELAY); //даем человеку успеть увидеть
        }
    } while (true);
}
```

Визуализатор

```
abstract class Visualizer
{
    public abstract void Draw(Board board);
}

class ConsoleVisualizer : Visualizer
{
   public override void Draw(Board board)
   {
       Console.Clear();
       for (int i = 1; i <= board.GetMaxX(); i++)
           for (int j = 1; j <= board.GetMaxY(); j++)
           {
               Console.CursorLeft = i; Console.CursorTop = j;
               GameObject obj = board.GetObjAt(i, j);
               if (obj == null) Console.Write('O’);
               if (obj != null && !obj.IsAlive()) Console.Write('x');
               if (obj != null && obj.IsAlive() && obj is Unit) Console.Write('W’);
           }
   }
}
```

Точка входа - Program.cs

```
class Program
{
    static void Main(string[] args)
    {
        Game game =  new Game();
        game.InitModel();
        game.RunModel();
        Console.ReadLine();
    }
}
```


X= 8, Y=8, HP = 200, side = Red
X= 4, Y=4, HP = 200, side = Blue
X= 7, Y=7, HP = 200, side = Red
X= 5, Y=5, HP = 200, side = Blue
X= 6, Y=6, HP = 200, side = Red
X= 5, Y=5, HP = 200, side = Blue
X= 6, Y=6, HP = 157, side = Red
X= 5, Y=5, HP = 152, side = Blue
X= 6, Y=6, HP = 115, side = Red
X= 5, Y=5, HP = 104, side = Blue
X= 6, Y=6, HP = 68, side = Red
X= 5, Y=5, HP = 63, side = Blue
X= 5, Y=5, HP = 6, side = Blue
Время в модели
Выжил один

Дальнейшее развитие модели

- Новые юниты
- Новые игровые объекты
- Интерактивность
- Графический интерфейс и прочие красоты

Дальнейшее развитие модели

- Новые юниты
  - Строитель – создается по аналогии с воином
    - Отличается инициализация базовых параметров в конструкторе (здоровье, сила, и т.п.)
    - Отличается перегрузка Action
    - Может порождать новые объекты
      - извещает игровой движок о необходимости такого создания
  - Дистанционный юнит (Стрелок)
    - Отличается инициализация базовых параметров в конструкторе (здоровье, сила, и т.п.)
    - Отличается перегрузка Action
    - Приведет к необходимости полноценной реализации видимости клеток и атаке не только соседних
    - Хороший повод добавить проверки в сам движок, которые не позволят Воину атаковать дистанционно, позволяя Стрелку

Дальнейшее развитие модели

- Новые игровые объекты
  - Защитные стены
    - Функциональность порождения новых объектов Строителем.
    - Приведут к необходимости создания алгоритма обхода препятствий юнитами
    - Либо расчета пути (с учетом видимости) и принятия решения, стоит ли обойти, или быстрее сломать стену
  - Дома
    - Приведут к необходимости расширения класса Колония для учета жилплощади
    - Возможность генерации юнитов Домами (раз в какое-то время, например)
    - Механизм голодания (снижение характеристик юнитов)
  - Игровые усиления (бонусы, powerups)
    - Собираемые на игровом поле предметы, меняющие характеристики юнитов («анти-голод»)
    - Приведут к необходимости модификации AI юнитов для их обнаружения и использования.

Дальнейшее развитие модели

- Интерактивность
  - Вариант одного персонажа
    - Порождаем ControllableUnit : Unit
    - перегружаем ControllableUnit.Action так, чтобы обрабатывать ввод от пользователя
  - Вариант множества управляемых персонажей
    - Пошаговая стратегия
      - В главном цикле вместо паузы, ждем ввода от пользователя координат каждому юниту
      - Используем введенные координаты в Unit.SetTargetLocation
      - Расширяем управление так, чтобы передавать не только координаты, но и любые команды (паттерн Команда, очередь команд, рассмотрим через пару лекций).
    - Стратегия реального времени
      - Параллельно с обработкой модельного мира обрабатываем UI-поток, и команды из него транслируем объектам модели

Дальнейшее развитие модели

- Графический интерфейс
  - Простейший вариант
    - Новый визуализатор, работающий в графическом режиме
    - Класс-контроллер, принимающий ввод от пользователя
      - Пулл-режим (Pull) – к нему обращается юнит, когда нуждается в команде
      - Пуш-режим (Push) – контроллер сам меняет состояние объектов модели
  - Продвинутый вариант
    - отдельная реализация графического интерфейса,
    - со своей объектной моделью визуальных компонент
    - возможно, работающая в отдельном потоке
    - возможно, на базе третьестороннего графического движка
    - отсылающая изменения в нашу модель, и получающая от нее обновления (например, с помощью механизма событий, который рассмотрим чуть позже)

Вопросы?