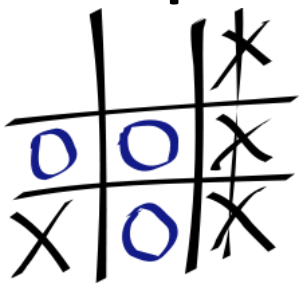
Объектно-ориентированное проектирование

Анализ и Проектирование. Пример

Анализ и Проектирование. Пример

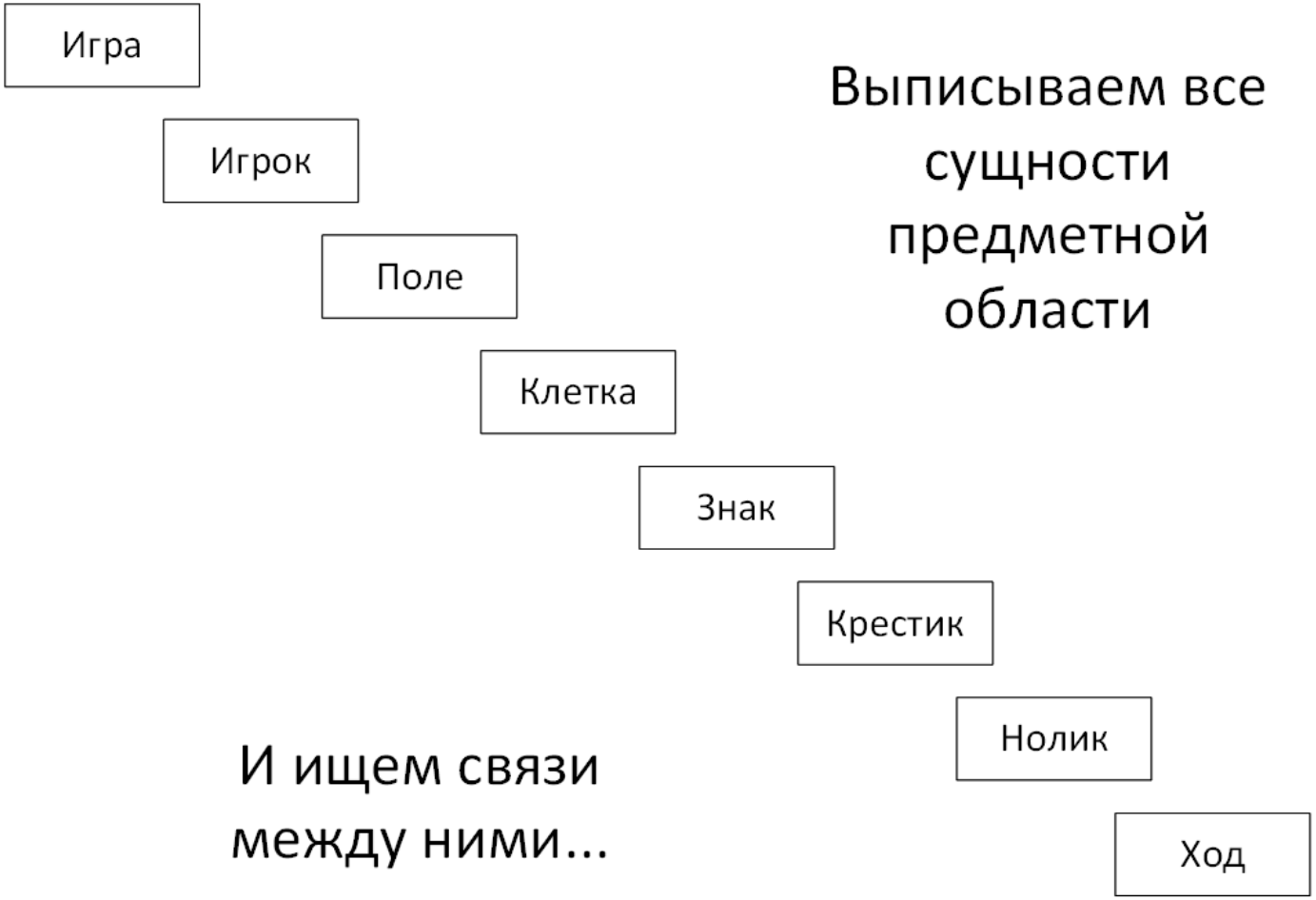


- Пример: игра в Крестики-нолики.
- Описание игры (https://ru.wikipedia.org/wiki/Крестики-нолики):
  - логическая **игра** между двумя **противниками** на квадратном **поле** 3 на 3 **клетки**. Один из **игроков играет** «**крестиками**», второй — «**ноликами**».
  - **Игроки** поочередно **ставят** на свободные **клетки поля** 3x3 **знаки** (один всегда **крестики**, другой всегда **нолики**). Первый, выстроивший в ряд 3 своих **фигуры** вертикально, горизонтально или диагонально, **выигрывает**. Первый **ход делает игрок**, ставящий **крестики**.

Выделение словаря предметной области

- Существительные
  - **Игра**
  - **Противник, Игрок**
  - **Поле**
  - **Клетка**
  - **Знак, Фигура**
  - **Крестик,**
  - **Нолик**
  - **Ход**
- Глаголы
  - **Играть**
  - **Ставить знаки, Делать ход**

Построение модели предметной области. Объекты.



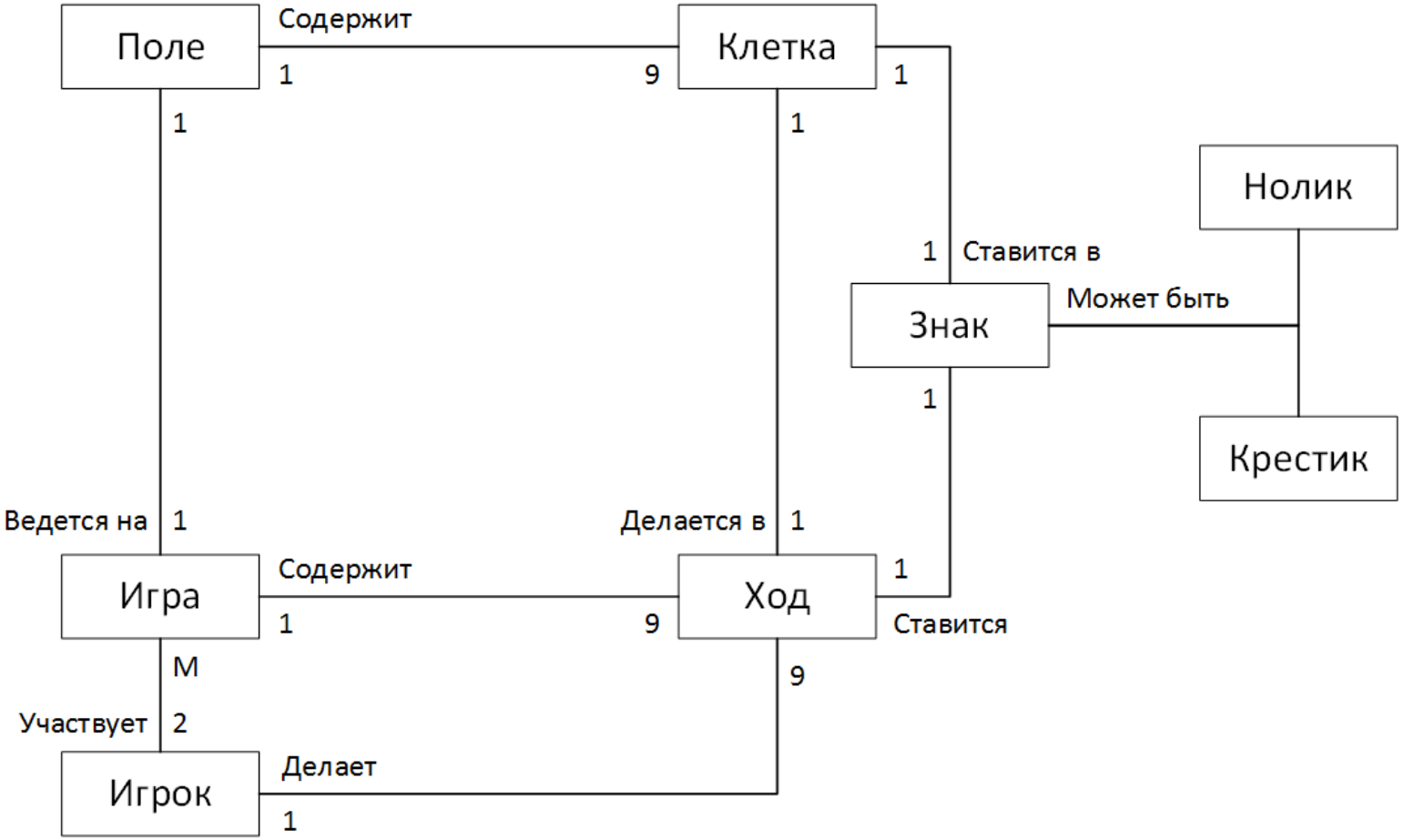
Выписываем все сущности предметной области

И ищем связи между ними...

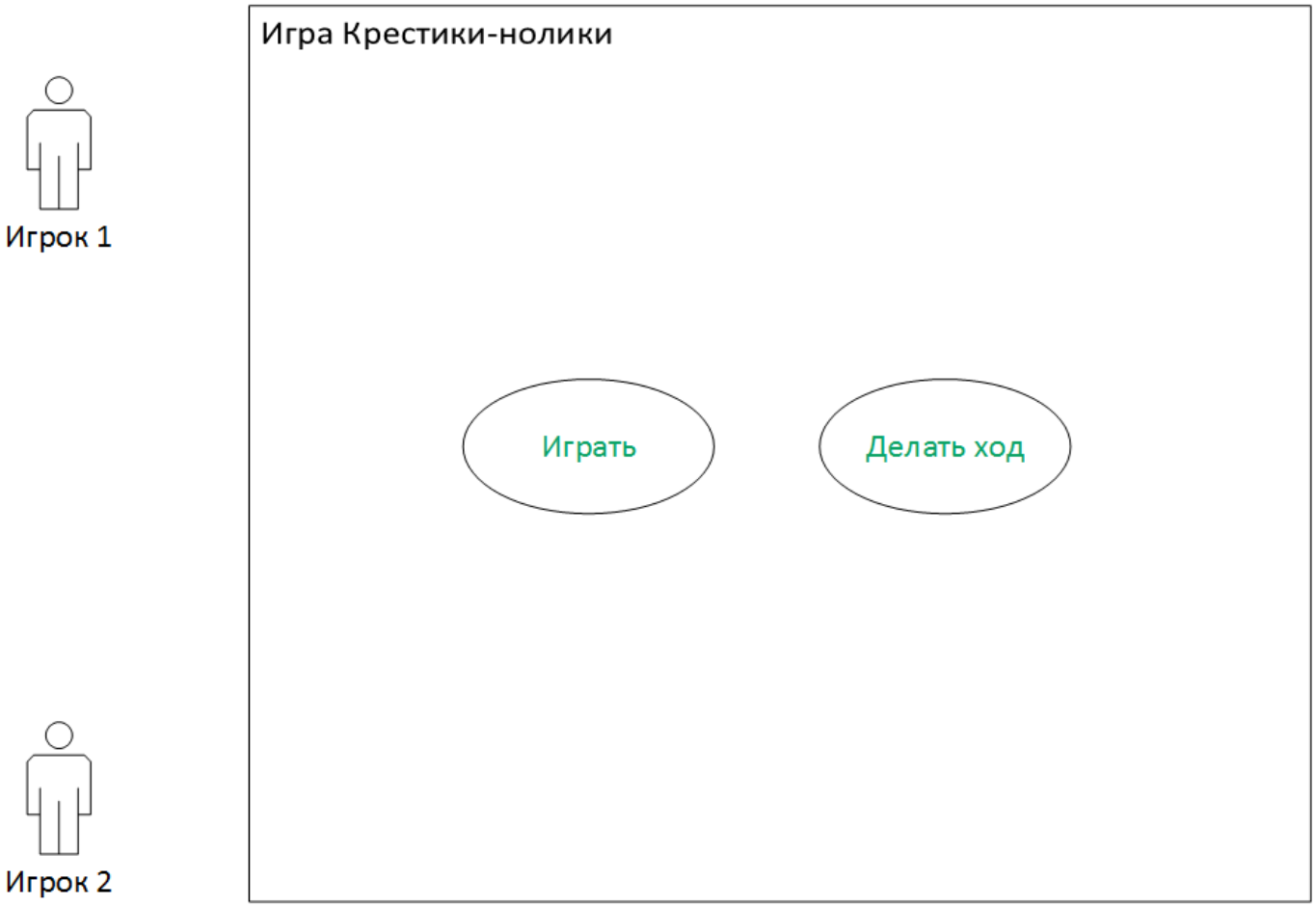
Построение модели предметной области. Отношения.

И так далее...

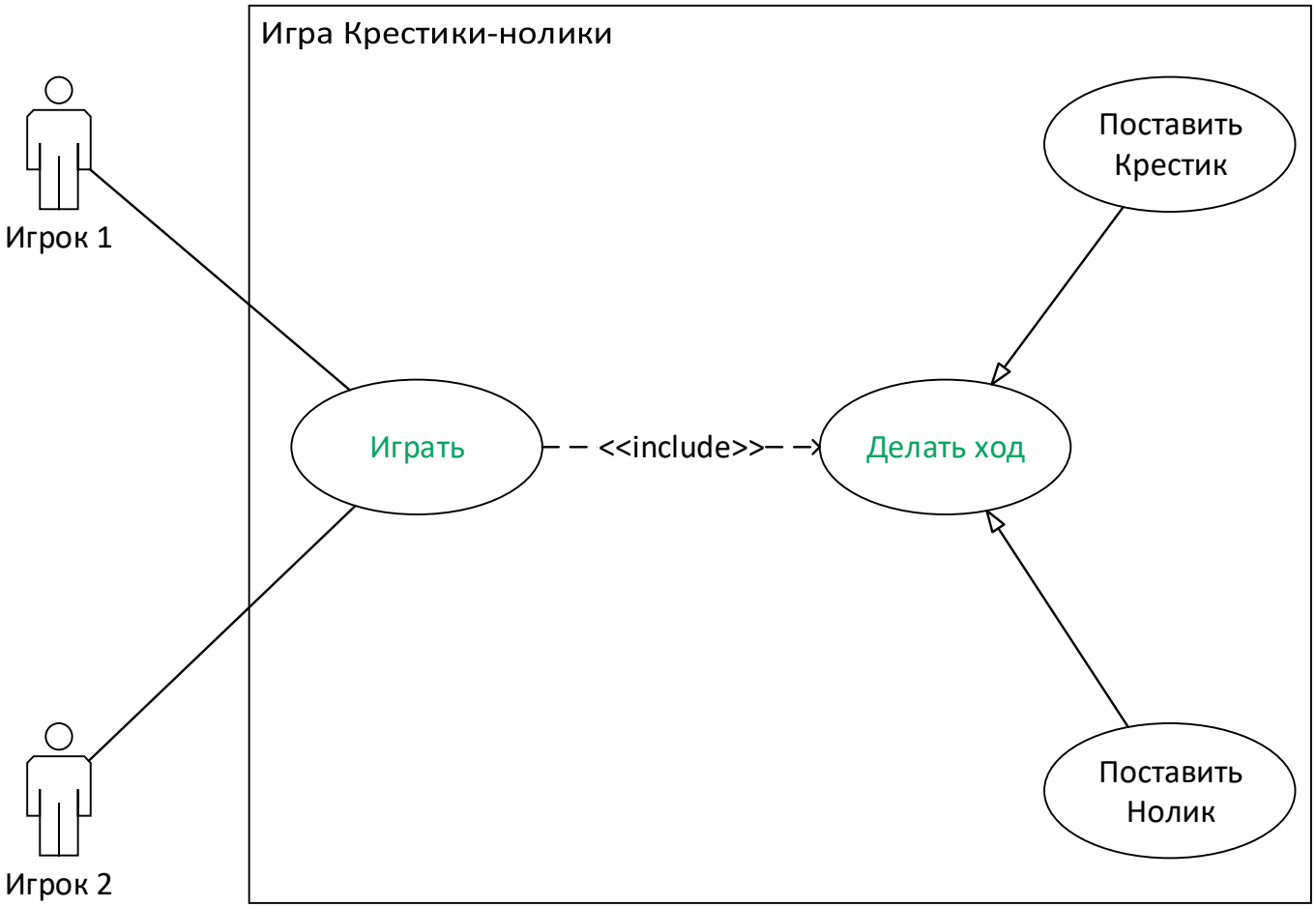
Построение модели предметной области. Диаграмма классов.



Определение вариантов использования (use cases).



Определение вариантов использования (use cases).



Определение вариантов использования (use cases).

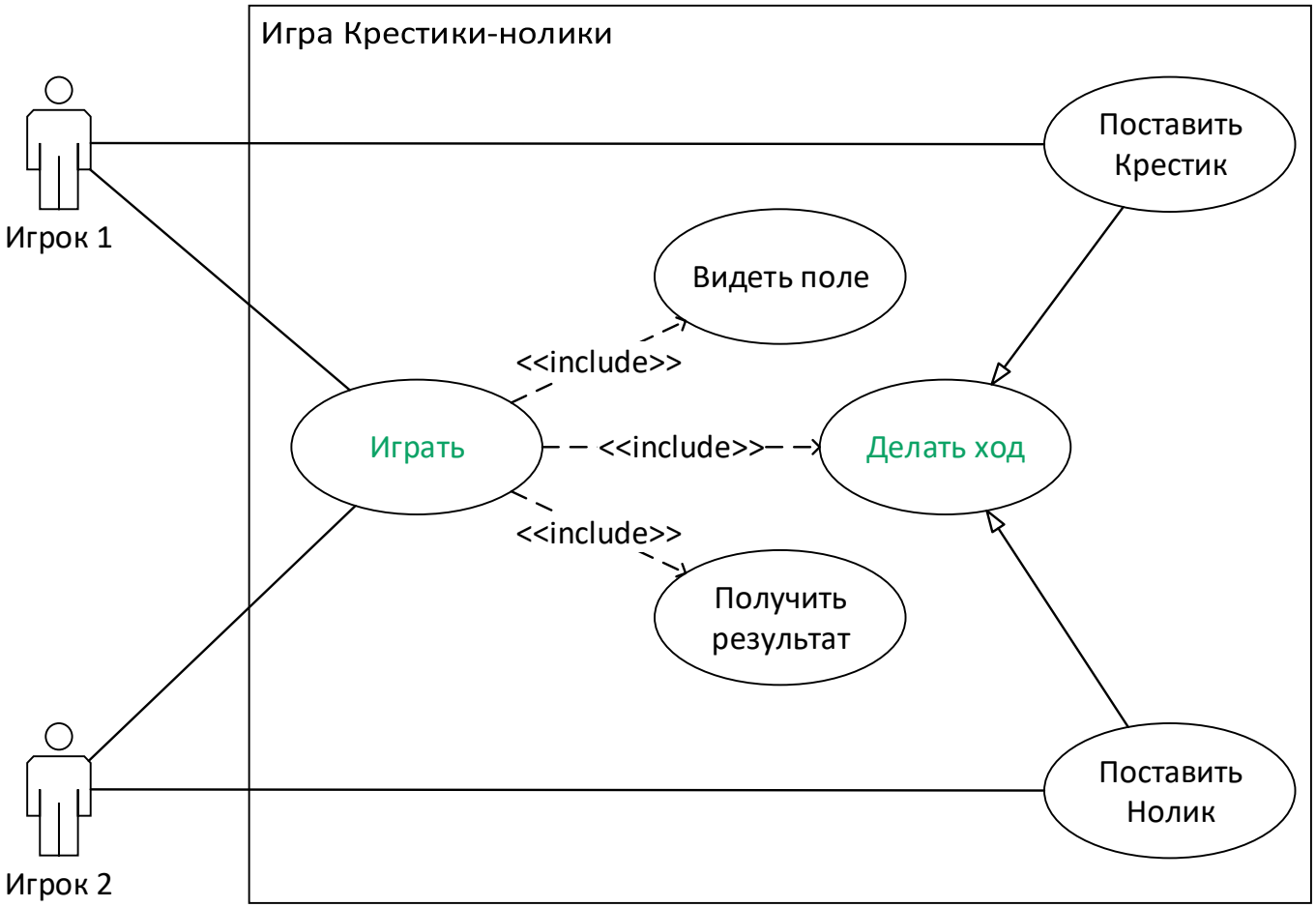


Диаграмма взаимодействия. Начало.

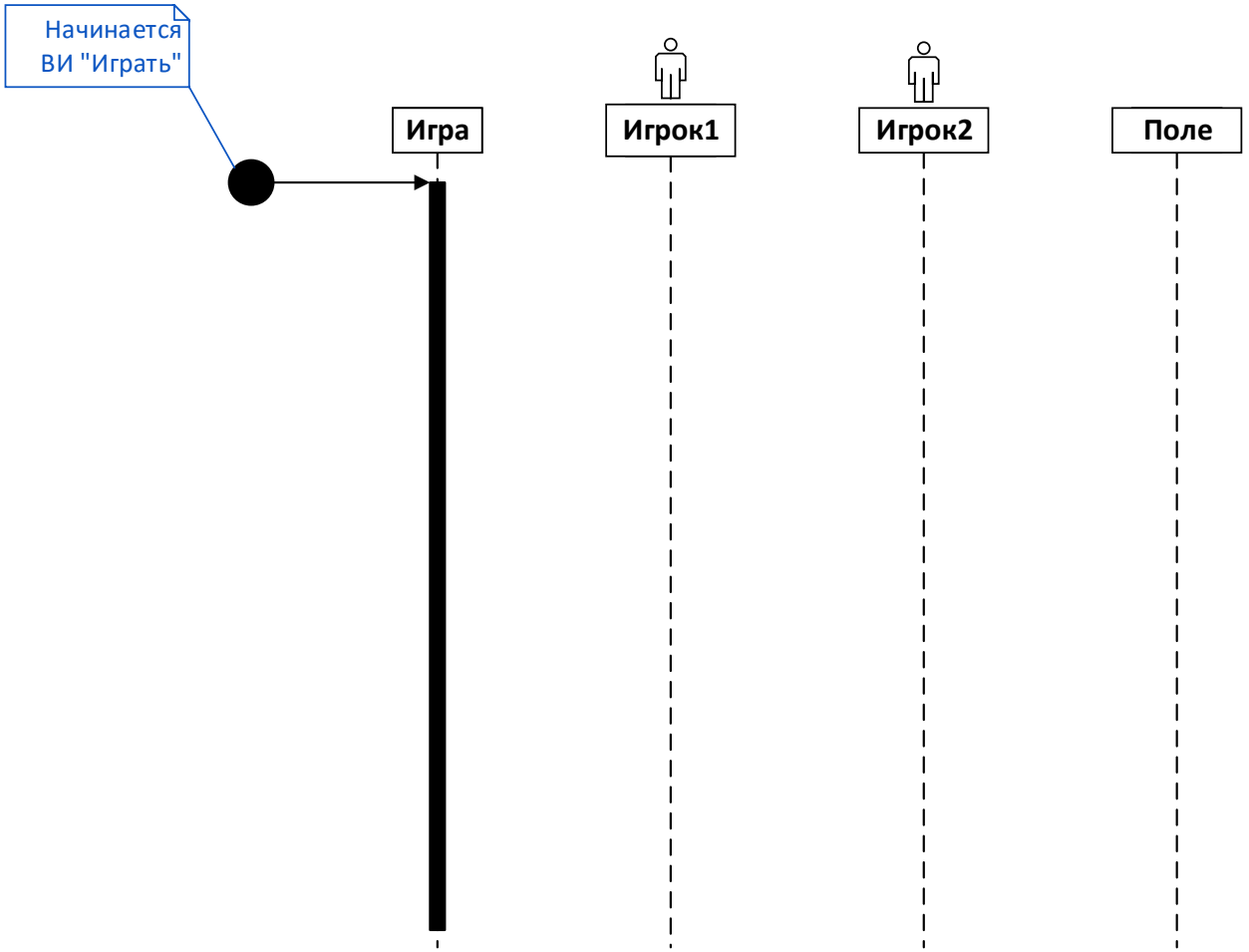


Диаграмма взаимодействия. Продолжение.

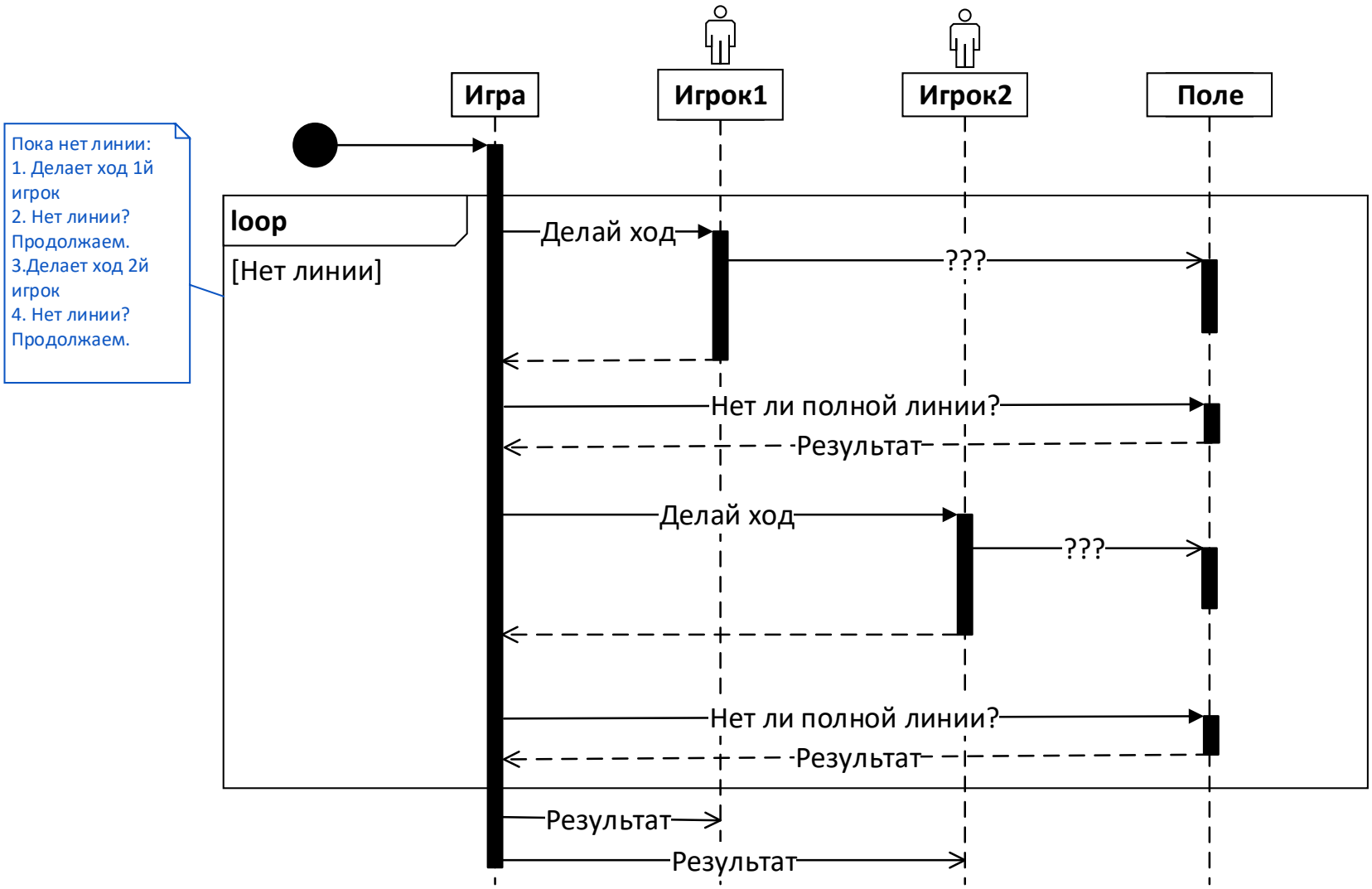


Диаграмма взаимодействия. Финальная версия.

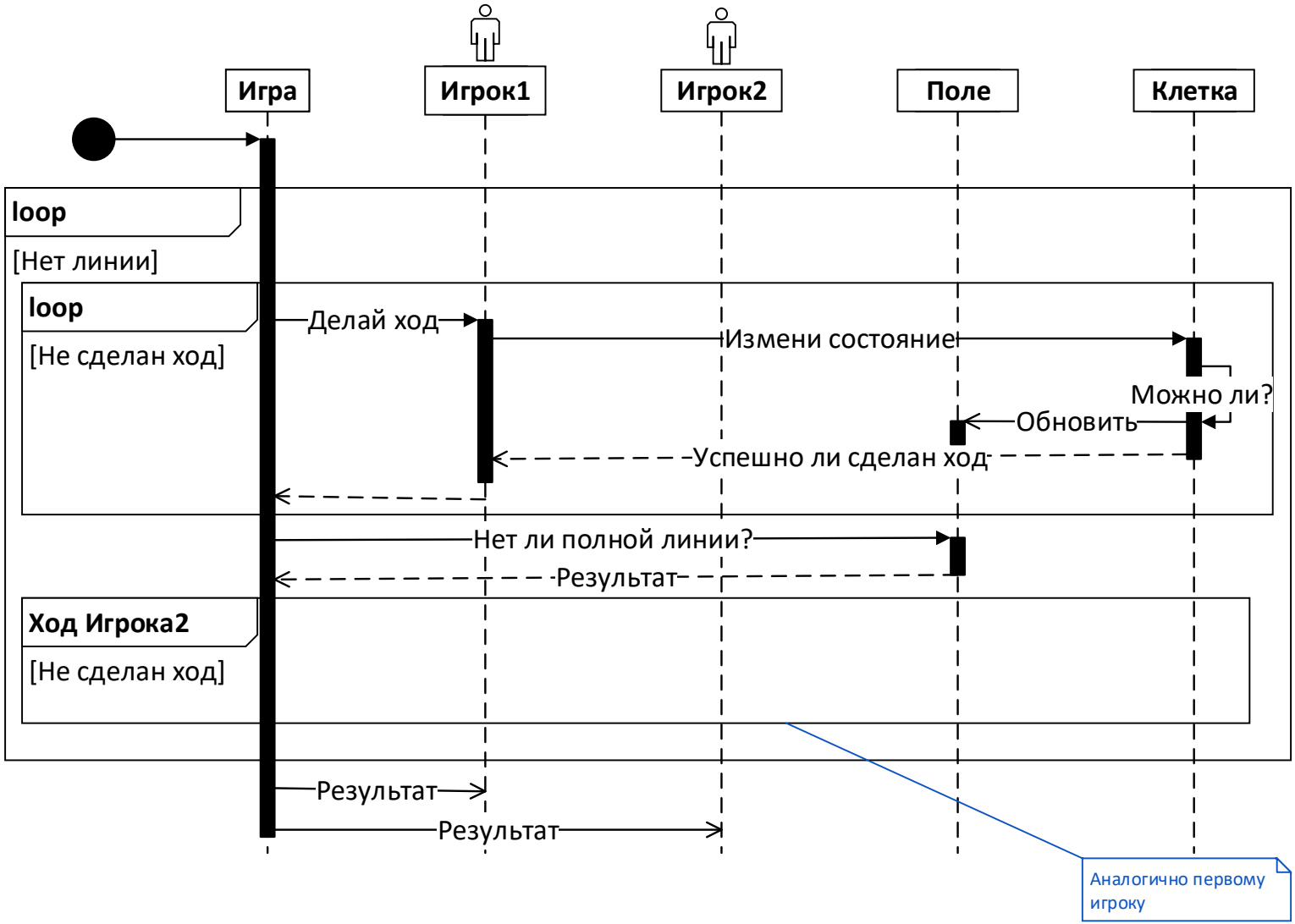


Диаграмма классов. Классы.

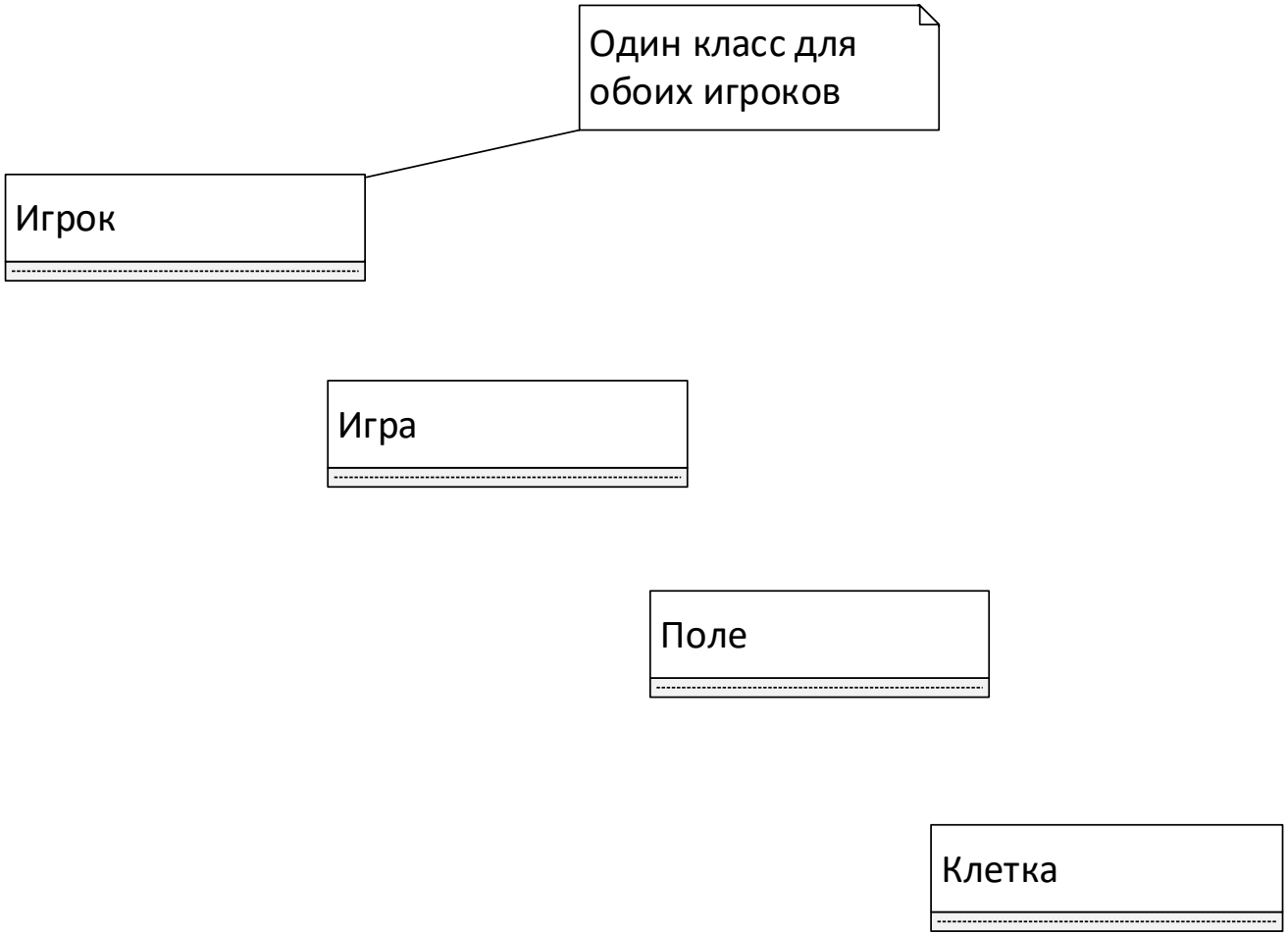


Диаграмма классов. Связи.

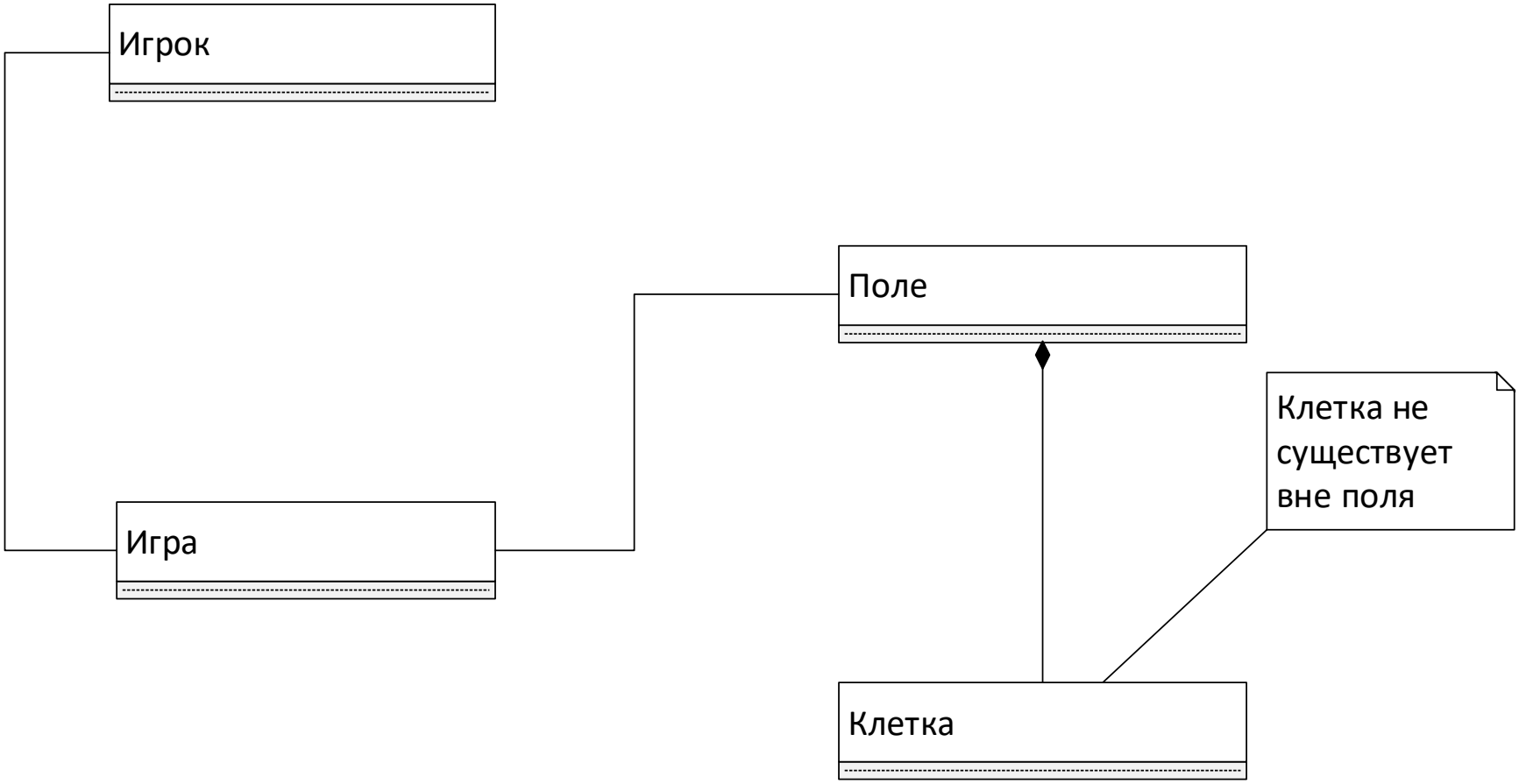


Диаграмма классов. Свойства и связанные с ними методы.

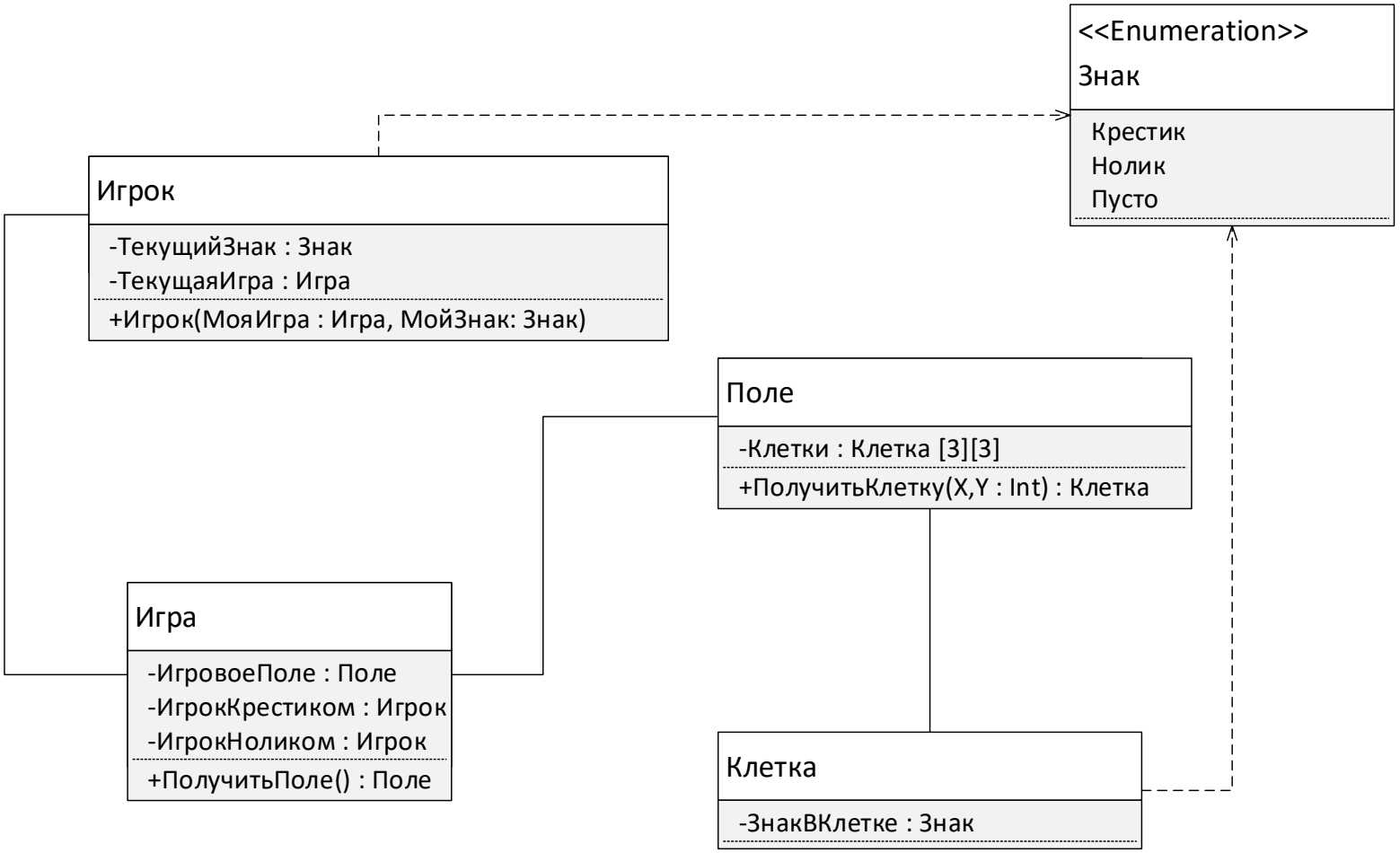
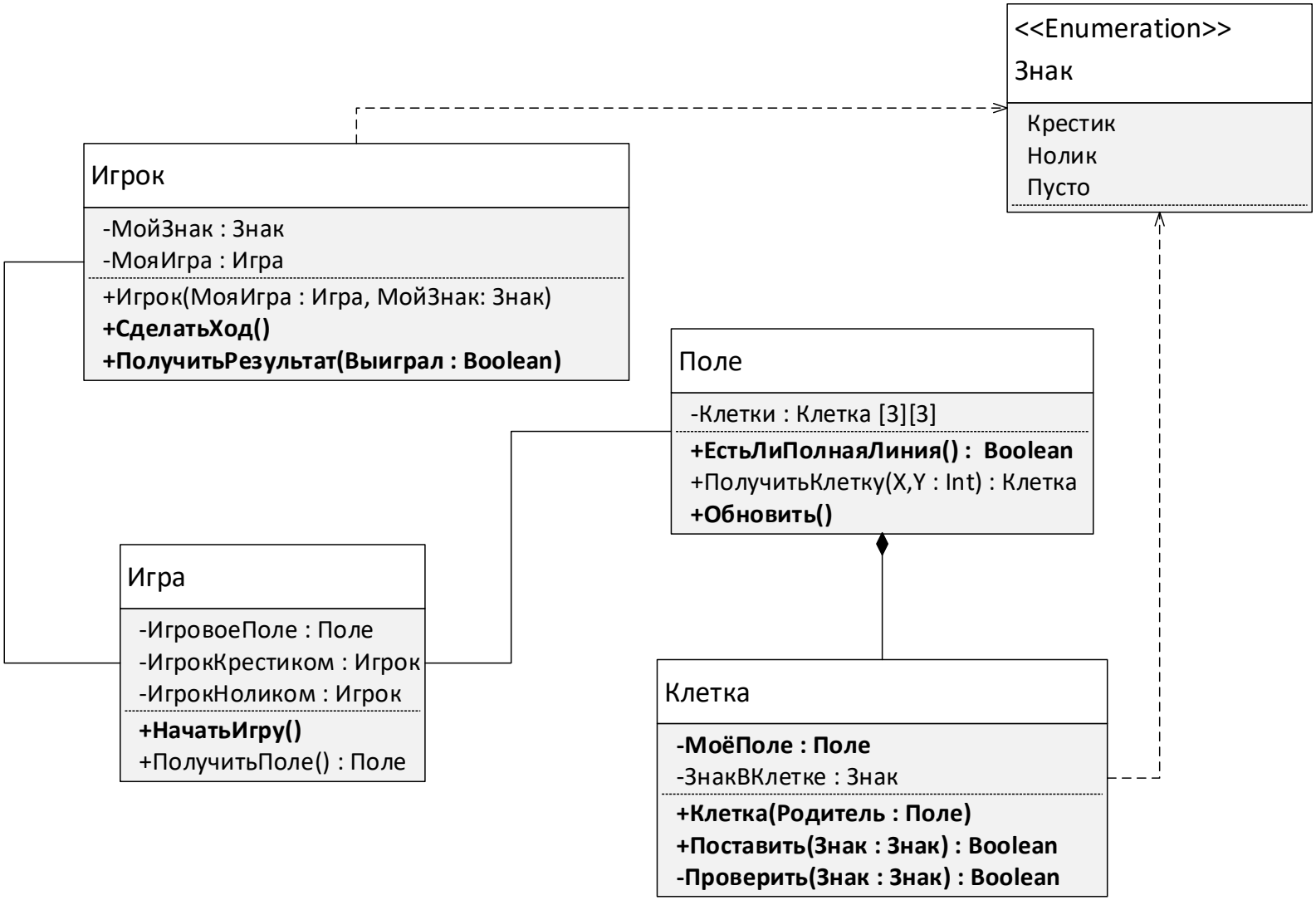


Диаграмма классов. Методы.



Реализация классов. Класс Игра

```
public void НачатьИгру()
{
        //инициализация
        Поле игровоеПоле = new Поле();
        Игрок игрокКрестиком = new Игрок(this, Знак.Крестик);
        Игрок игрокНоликом = new Игрок(this, Знак.Нолик);
        Игрок текущийИгрок = игрокКрестиком; //Крестик ходит первым

        do  //основной цикл игры
        {
                текущийИгрок.СделатьХод();
                текущийИгрок = текущийИгрок == игрокКрестиком ?
                        игрокНоликом : игрокКрестиком //меняем текущего
        } while ( !игровоеПоле.ЕстьЛиПолнаяЛиния() );

        //сообщаем  игрокам результат – код пропущен для краткости
}
```

Реализация классов. Класс Игрок

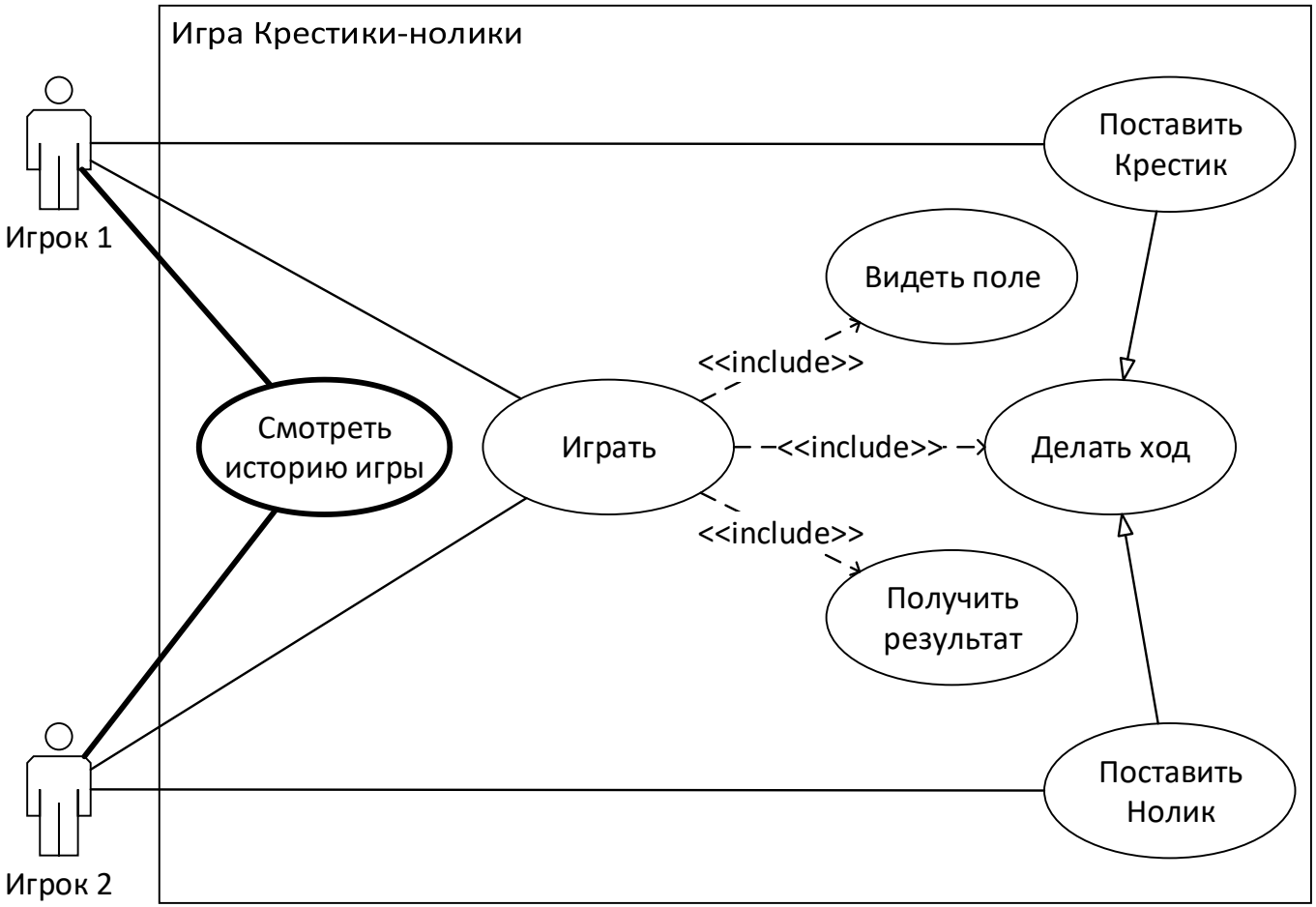
```
public void СделатьХод()
{
        bool success;

        do
        {
                //пользователь вводит клетку, в которую он делает ход
                int X = ПолучитьОтПользователяX();
                int Y = ПолучитьОтПользователяY();

                 //пробуем сделать ход в эту клетку
                success = ТекущаяИгра.ПолучитьПоле().
                        ПолучитьКлетку(X,Y).Поставить(МойЗнак));
        } while (!success); //пока не получится
}
```

Реализация классов. Класс Клетка

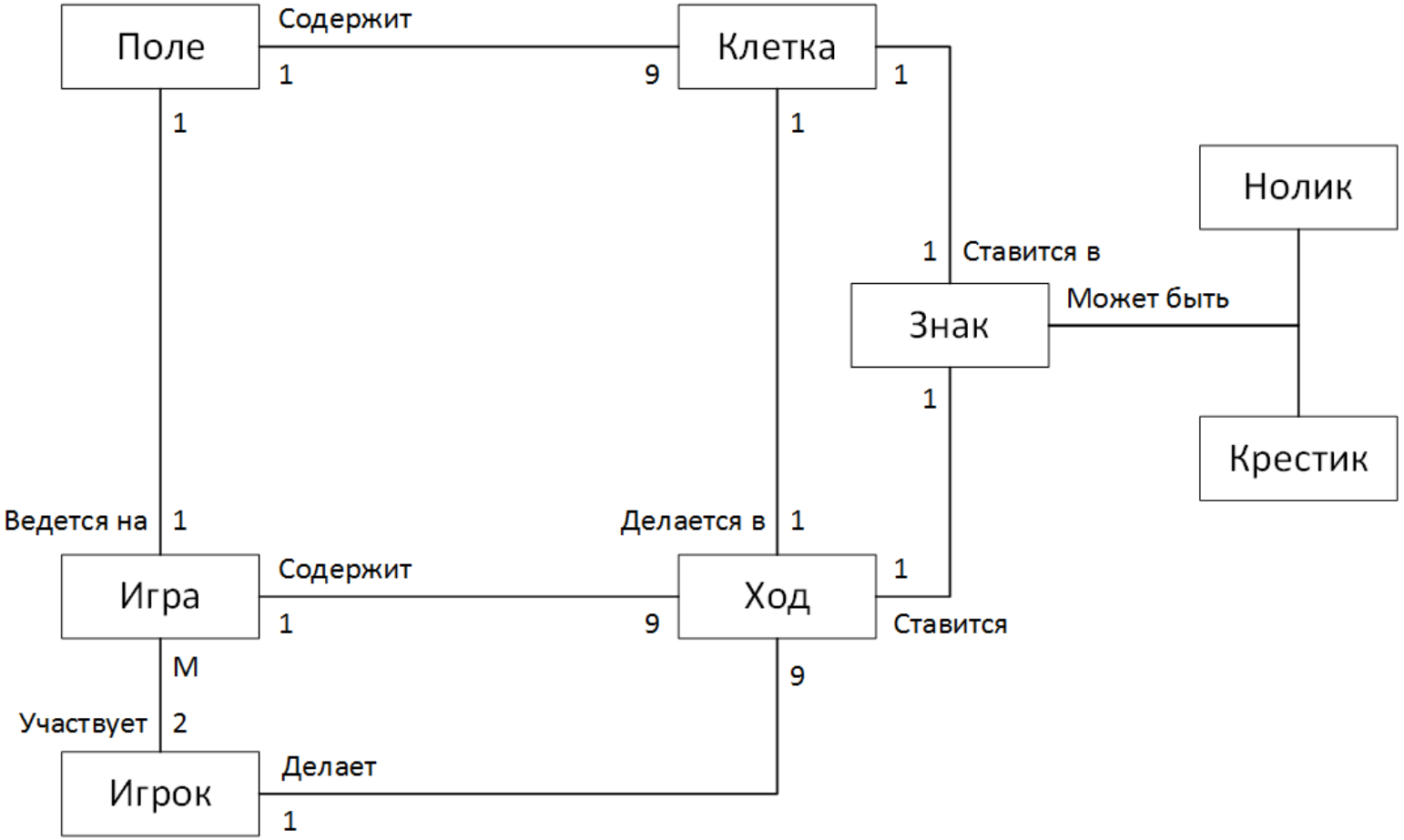
```
public bool Поставить(новыйЗнак : Знак)
{
        if (Проверить(новыйЗнак)) //если разрешено ставить, то ставим
        {
                ЗнакВКлетке = новыйЗнак;
                МоёПоле.Обновить(); //вызываем перерисовку поля
                return true;
        }
        else return false; //иначе, говорим, что не смогли
}

private bool Проверить(новыйЗнак : Знак)
{
        //разрешаем ставить только в пустые клетки
        return ЗнакВКлетке == Знак.Пусто;
}
```

Изменения в модели

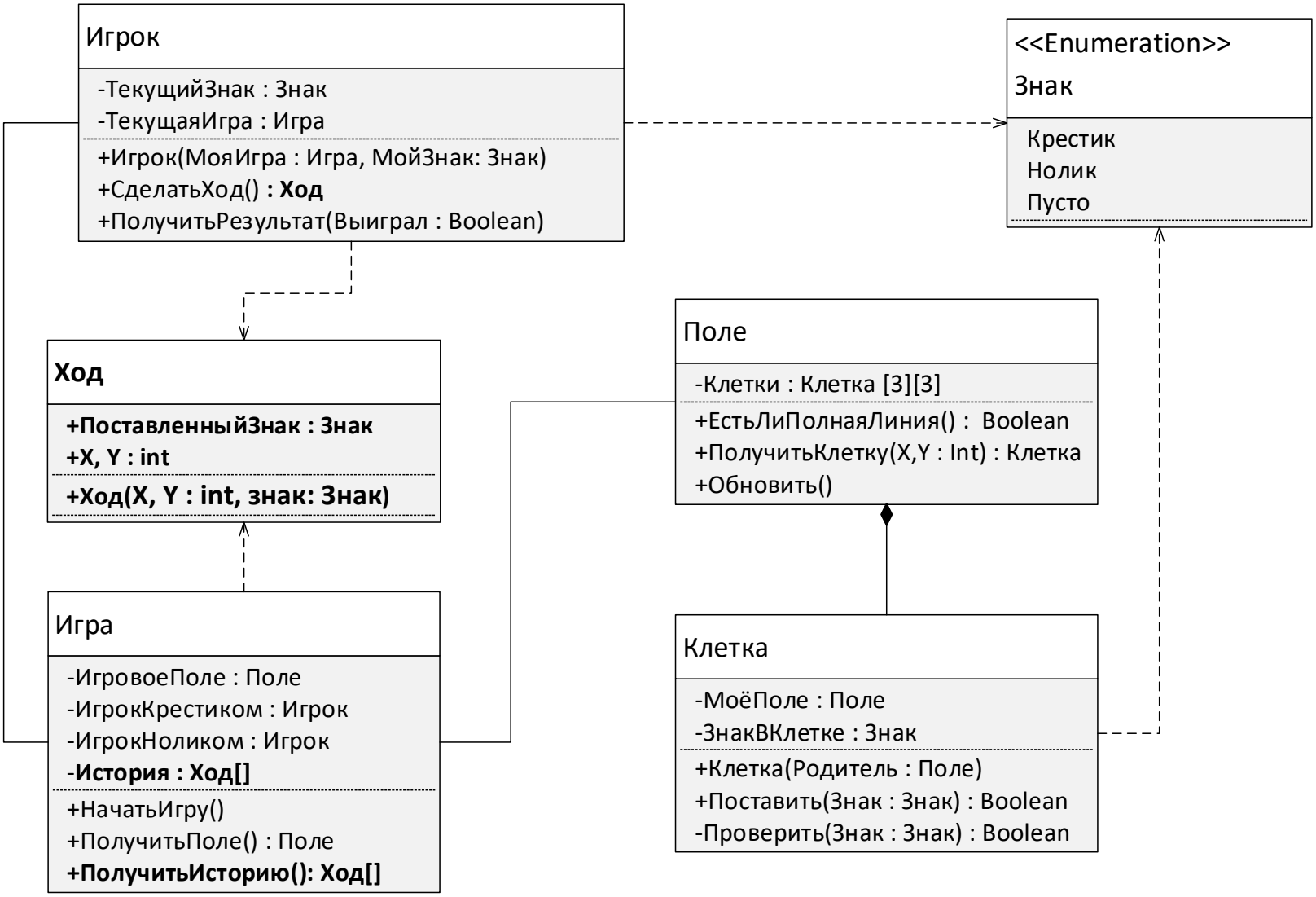
Расширяем сценарии использования



Вспоминаем модель предметной области



Расширяем диаграмму классов



Расширяем код классов

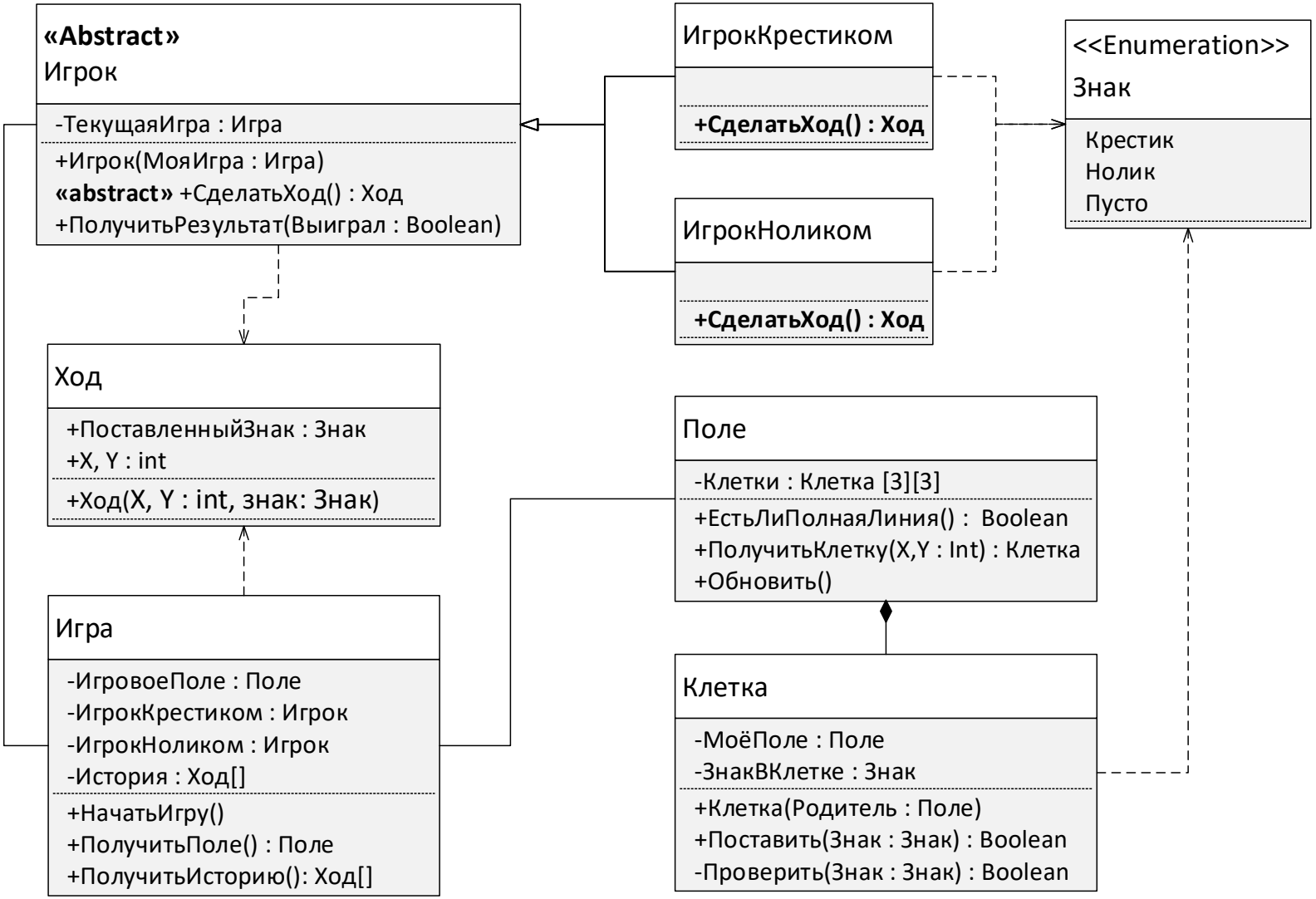
- Класс Игрок

```
public Ход СделатьХод()
{
        bool success;
        do
        {
                int X = ПолучитьОтПользователяX();
                int Y = ПолучитьОтПользователяY();
                success = ТекущаяИгра.ПолучитьПоле().
                        ПолучитьКлетку(X,Y).Поставить(МойЗнак));
        } while (!success);
        return new Ход(X, Y, МойЗнак);
}
```

- Класс Игра, метод НачатьИгру():

```
do  //основной цикл игры
{
        Ход новыйХод = ТекущийИгрок.СделатьХод();
        История.Добавить(новыйХод);
        СменитьТекущегоИгрока();
} while ( !ИгровоеПоле.ЕстьЛиПолнаяЛиния() );
```

А как же наследование и полиморфизм?



Класс Игра (вариант с отдельными классами Игроков)

```
public void НачатьИгру()
{
        //инициализация
        Поле игровоеПоле = new Поле();

        //убрали зависимость от энума
        ИгрокКрестиком крестик = new ИгрокКрестиком(this);
        ИгрокНоликом нолик = new ИгрокНоликом(this);
        Игрок текущийИгрок = крестик; //Крестик ходит первым

        do  //основной цикл игры
        {
                текущийИгрок.СделатьХод();
                текущийИгрок = текущийИгрок == крестик ?
                        нолик : крестик //меняем текущего
        } while ( !ИгровоеПоле.ЕстьЛиПолнаяЛиния() );

        //сообщаем  игрокам результат – код пропущен для краткости
}
```

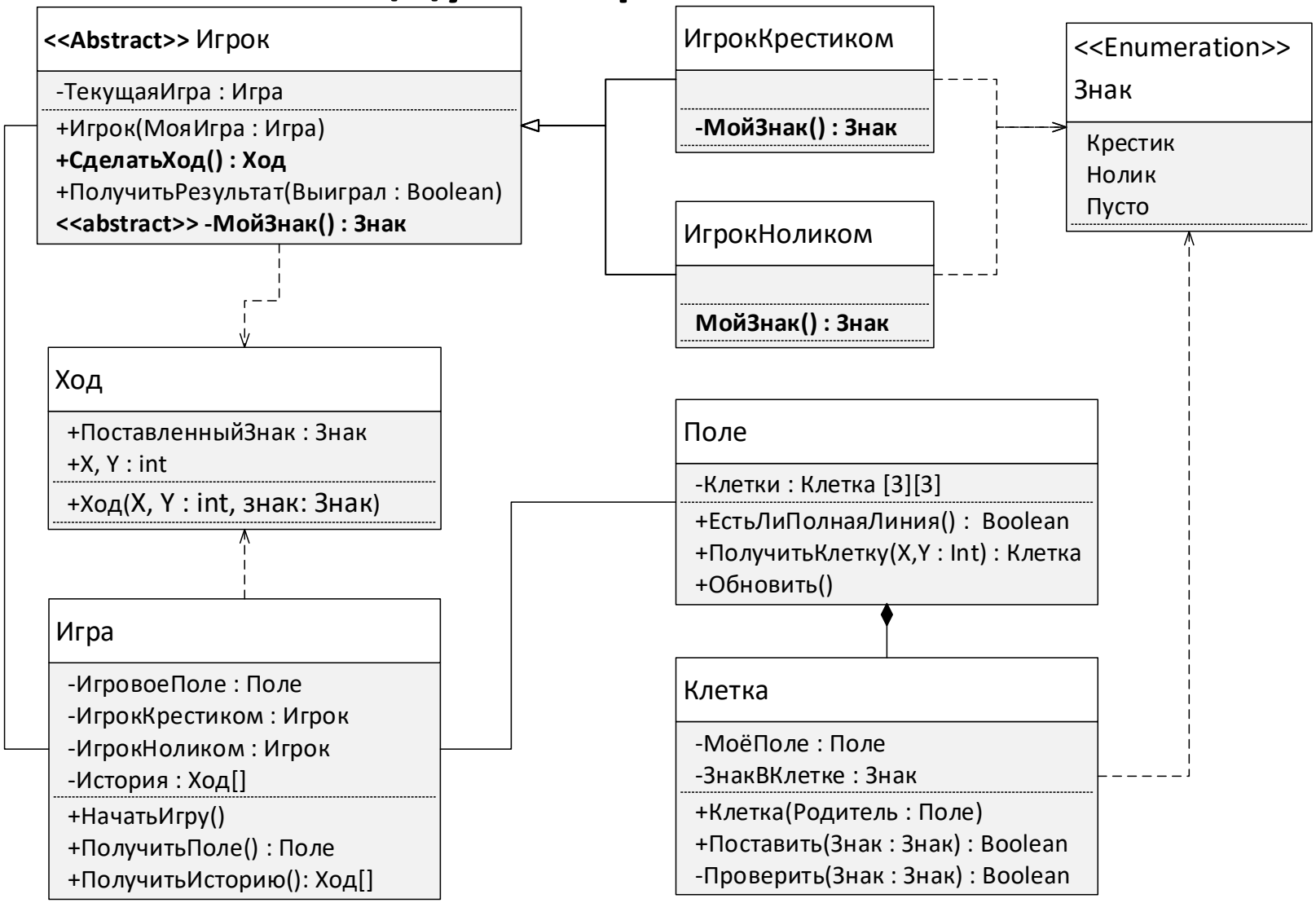
Неоптимальный вариант с наследованием (не делайте так)

- Класс ИгрокКрестиком

```
public Ход СделатьХод()
{
        Знак мойЗнак = Знак.Крестик;
        bool success;
        do
        {
                int X = ПолучитьОтПользователяX();
                int Y = ПолучитьОтПользователяY();
                success = ТекущаяИгра.ПолучитьПоле().
                        ПолучитьКлетку(X,Y).Поставить(мойЗнак));
        } while (!success);
        return new Ход(X, Y, мойЗнак);
}
```

- Класс ИгрокНоликом – аналогичен. **Повторение кода – это плохо.**

Улучшаем дизайн – убираем дублирование



Оптимизированный вариант с наследованием.

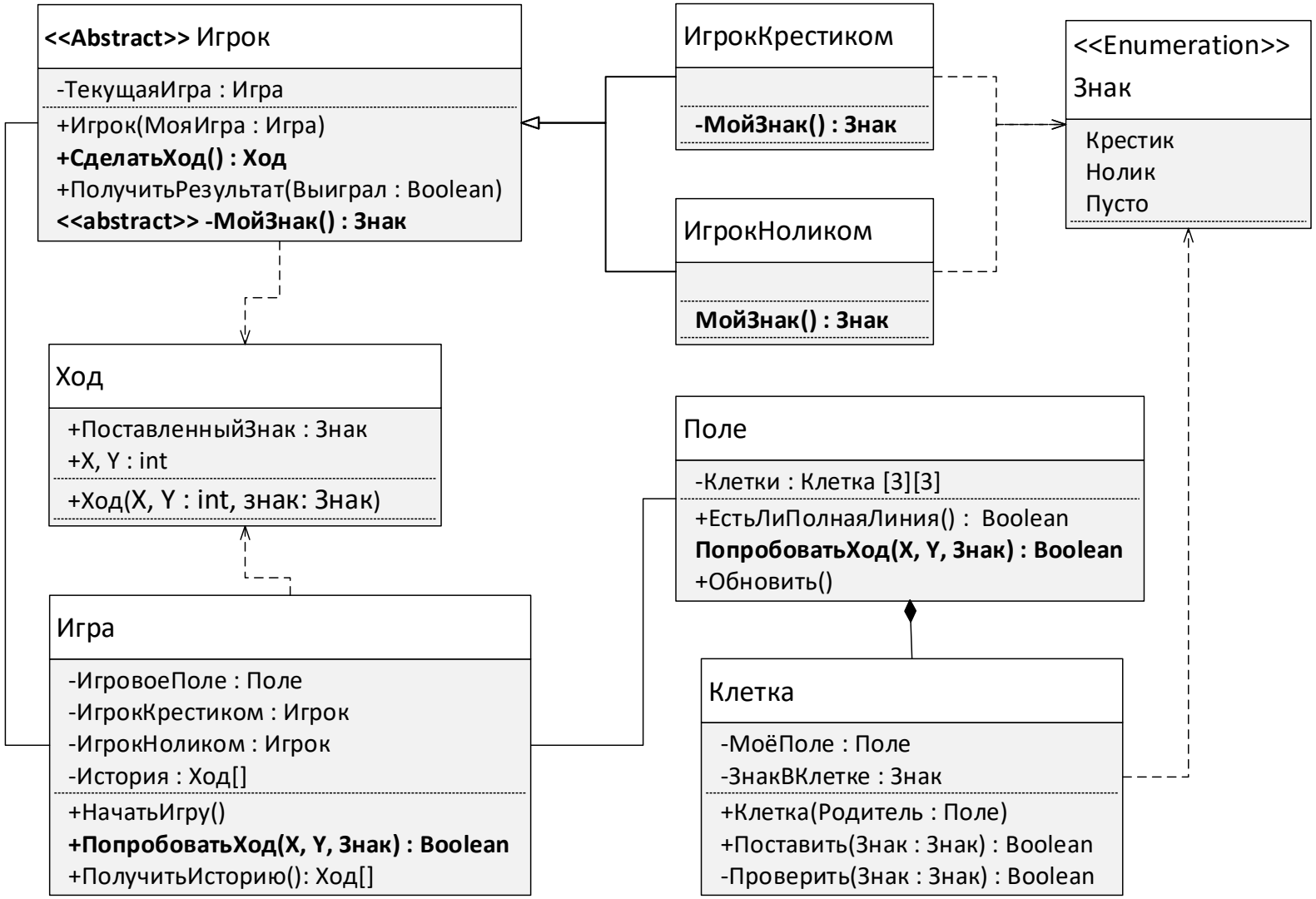
- Абстрактный класс Игрок

```
private Знак МойЗнак() =0; //чисто виртуальный метод

public Ход СделатьХод()
{
        Знак мойЗнак = МойЗнак() ;
        bool success;
        do
        {
                int X = ПолучитьОтПользователяX();
                int Y = ПолучитьОтПользователяY();
                success = ТекущаяИгра.ПолучитьПоле().
                        ПолучитьКлетку(X,Y).Поставить(мойЗнак));
        } while (!success);
        return new Ход(X, Y, мойЗнак);
}
```

- Класс ИгрокКрестиком

```
private Знак МойЗнак() { return Знак.Крестик; }
```

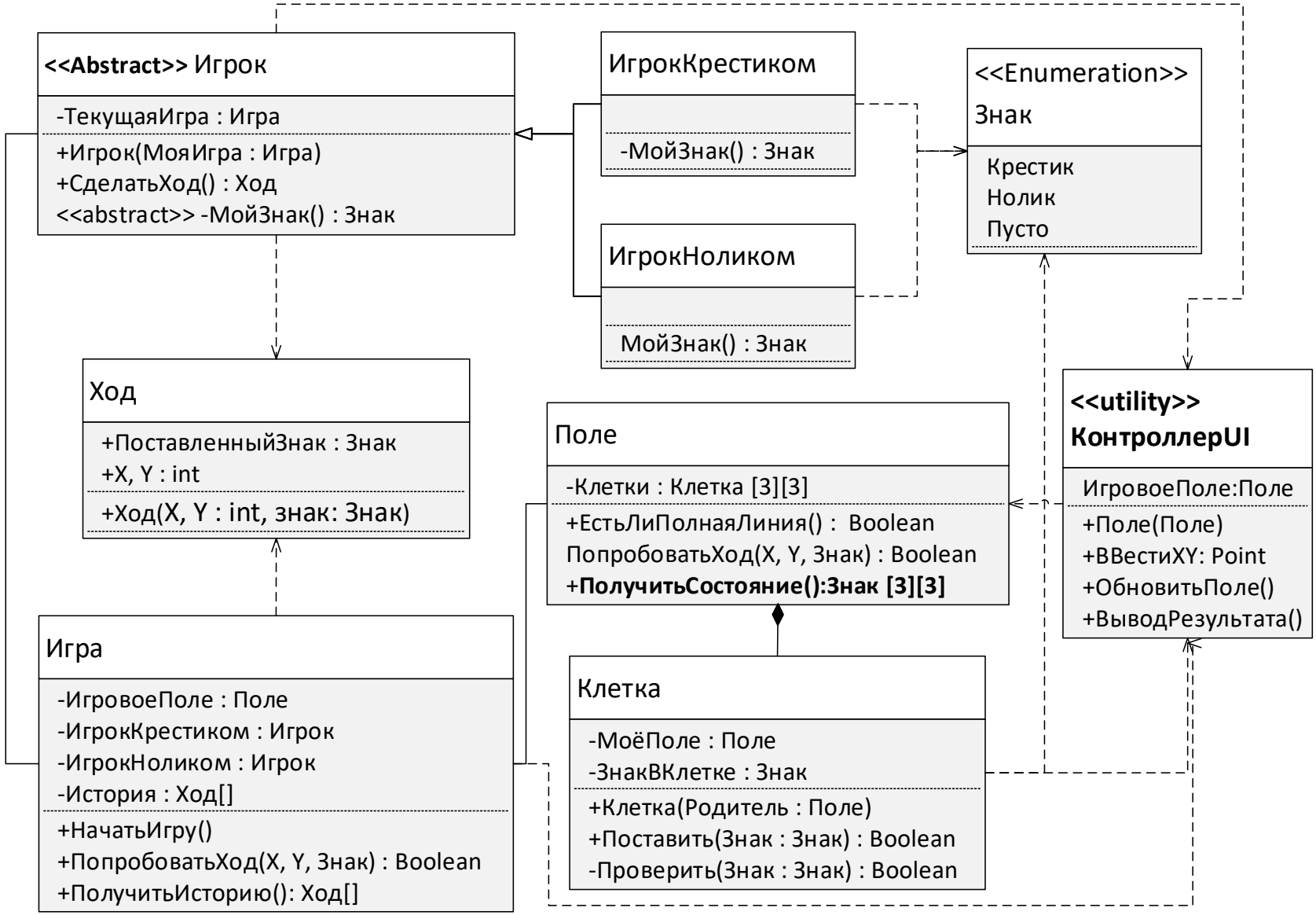
Убираем нарушение принципа Деметры



Убираем нарушение принципа Деметры

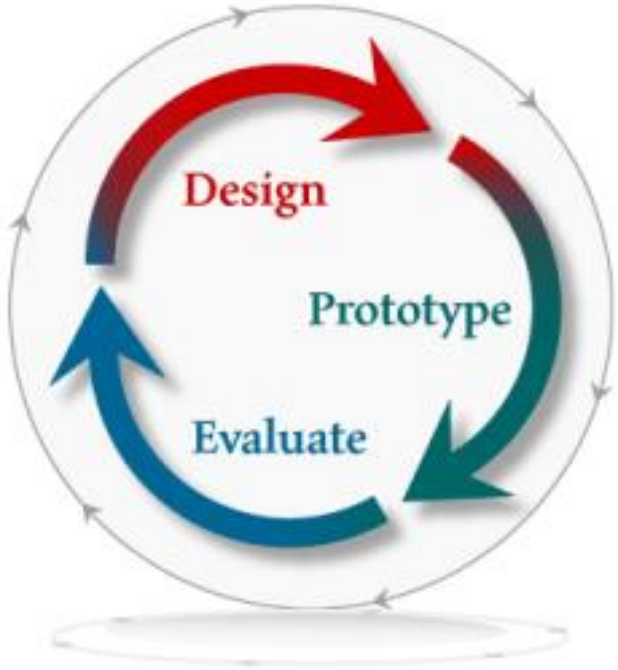
```
public Ход СделатьХод()
{
        Знак мойЗнак = МойЗнак() ;
        bool success;
        do
        {
                int X = ПолучитьОтПользователяX();
                int Y = ПолучитьОтПользователяY();
                /*
                success = ТекущаяИгра.ПолучитьПоле().
                                ПолучитьКлетку(X,Y).Поставить(мойЗнак));
                */
                success = ТекущаяИгра.ПопробоватьХод(X,Y,мойЗнак));
        } while (!success);
        return new Ход(X, Y, мойЗнак);
}
```

Убираем нарушение SRP



Итерационность процесса проектирования

- Единственно верного решения НЕТ. Как нет предела совершенству.
- Хорошая модель – модель готовая и к реализации, и к изменениям. Обычно это модель, построенная на основе предметной области.
- Модель улучшается до тех пор, пока она не станет удовлетворять всем требованиям к системе.



Когда остановиться?

- Когда модель становится реализуемой.
- Когда модель покрывает все функциональные и нефункциональные требования к разрабатываемой системе.
- Лучшее – враг хорошего (Overengineering или «ООПухоль мозга»)
- Помним, что инструменты ООП – средства, а не цель.