Poniżej zamieszczam fragment kodu, realizującego symulację górniczego miasta krasnoludów.

Postanowiłem zaprezentować klasę Hospital, w której realizowane są dwa różne zachowania w zależności od tury. Wykorzystuje ona strategię losowania rodzaju postaci, jak również strategię generowania szans narodzin nowej postaci. Ponadto zarządza fabryką nowych postaci. Uważam, że poniższa struktura jest łatwo testowalna, co potwierdzają dołączone na końcu dokumentu testy, ponieważ wykorzystując obiekty mock'owalne uzyskałem przewidywalne i powtarzalne wyniki. Kod jest także utrzymany w jednej konwencji nazewniczej.

```
public class Hospital
{
    private IIsDwarfBornRandomizer _isDwarfBornRandomizer;
    private IDwarfTypeRandomizer _dwarfTypeRandomizer;
    private DwarfFactory _dwarfFactory;
    private ILog _logger;
    private int _numberOfBirths;

    #region Contructors

    public Hospital(ILog logger = null, IIsDwarfBornRandomizer isDwarfBornRandomizer = null,
        IDwarfTypeRandomizer dwarfTypeRandomizer = null)
    {
        _isDwarfBornRandomizer = (isDwarfBornRandomizer != null)
            ? isDwarfBornRandomizer : new DwarfBornGenerationStrategy();
        _dwarfTypeRandomizer = (dwarfTypeRandomizer != null) ?
            dwarfTypeRandomizer : new DwarfTypeGenerationStrategy();
        _logger = (logger != null) ? logger : new Logger();
        _dwarfFactory = new DwarfFactory();
        _numberOfBirths = 0;
    }
    #endregion
```

Wadą tej klasy, którą dostrzegam jest niekoniecznie czytelny konstruktor, ze względu na wykorzystanie domyślnych parametrów. Można to zmienić wykorzystując dwa oddzielne konstruktory, jeden przeznaczony do testów, drugi z parametrami domyślnymi.

```
public void CreateDwarves(SimulationState simulationState)
{
    if(simulationState.turn == 1)
    {
        simulationState.dwarves.AddRange(CreateInitialDwarves());
    }
    else
    {
        simulationState.dwarves.AddRange(CreateSingleRandomDwarf());
    }
}
private List<IDwarf> CreateInitialDwarves()
{
    List<IDwarf> dwarves = new List<IDwarf>();
    for(int i = 0; i < 10; i++)
    {
        var type = _dwarfTypeRandomizer.GiveMeDwarfType(omitSuicider: true);
        dwarves.Add(_dwarfFactory.Create(type));
    }
    _logger.AddLog("10 dwarves have been born.");
    return dwarves;
}
```

```
private List<IDwarf> CreateSingleRandomDwarf()
{
    List<IDwarf> dwarves = new List<IDwarf>();
    if(_isDwarfBornRandomizer.IsDwarfBorn())
    {
        var type = _dwarfTypeRandomizer.GiveMeDwarfType(omitSuicider: false);
        dwarves.Add(_dwarfFactory.Create(type));
        _logger.AddLog($"{type} has been born.");
        _numberOfBirths++;
    }
    return dwarves;
}

public int GetNumberOfBirths()
{
    return _numberOfBirths;
}
```

Mimo prywatnych metod CreateInitialDwarves oraz CreateSingleRandomDwarf są one nadal testowalne stosując odpowiednie zachowania klas losujących i ustawienia odpowiednich wartości obiektu SimulationState. Poniższe testy są także dowodem tej tezy.

```
internal class HospitalDwarfBirthTests
{
    private Mock<IIsDwarfBornRandomizer> isDwarfBornMock;
    private Mock<IDwarfTypeRandomizer> dwarfTypeMock;
    private Hospital hospital;
    [SetUp]
    public void Setup()
    {
        isDwarfBornMock = new Mock<IIsDwarfBornRandomizer>();
        isDwarfBornMock.Setup(x => x.IsDwarfBorn(100)).Returns(true);
        dwarfTypeMock = new Mock<IDwarfTypeRandomizer>();
        dwarfTypeMock.Setup(x => x.GiveMeDwarfType(true)).Returns(DwarfType.Father);
    }
```

```
[Test]
public void T100_ShouldBuild10DwarfFathersWhenTurnEquals1()
{
    //given
    SimulationState simulationState = new SimulationState();
    hospital = new Hospital(new Logger(),isDwarfBornMock.Object, dwarfTypeMock.Object);
    Assert.IsTrue(simulationState.dwarves.Count == 0);
    //when
    hospital.CreateDwarves(simulationState);
    //then
    Assert.IsTrue(simulationState.dwarves.Count == 10);
    foreach (var dwarf in simulationState.dwarves)
    {
        Assert.IsTrue(dwarf._dwarfType == DwarfType.Father);
    }
}
```

```
[Test]
public void T101_ShouldBuild1SingleDwarfWhenTurnNotEquals1()
{
    //given
    SimulationState simulationState = new SimulationState();
    simulationState.turn = 2;
    dwarfTypeMock.Setup(x => x.GiveMeDwarfType(false)).Returns(DwarfType.Single);
    hospital = new Hospital(new Logger(), isDwarfBornMock.Object, dwarfTypeMock.Object);
    Assert.IsTrue(simulationState.dwarves.Count == 0);
    //when
    hospital.CreateDwarves(simulationState);
    //then
    Assert.IsTrue(simulationState.dwarves.Count == 1);
    Assert.IsTrue(simulationState.dwarves[0]._dwarfType == DwarfType.Single);
}
```

W powyższych metodach testujących dodałem asercje także przed wywołaniem badanej metody, aby mieć pewność zaistnienia zmian wewnątrz obiektu simulationState.