Akka.NET

Aufgabe 2: Zahlen raten

Single Responsibility

- ähnlich zu OOP:
  Ein Aktor ist für eine Aufgabe zuständig
  „do one thing – do it completely, do it well“
- Jede neue Aufgabe erfordert einen neuen Aktor

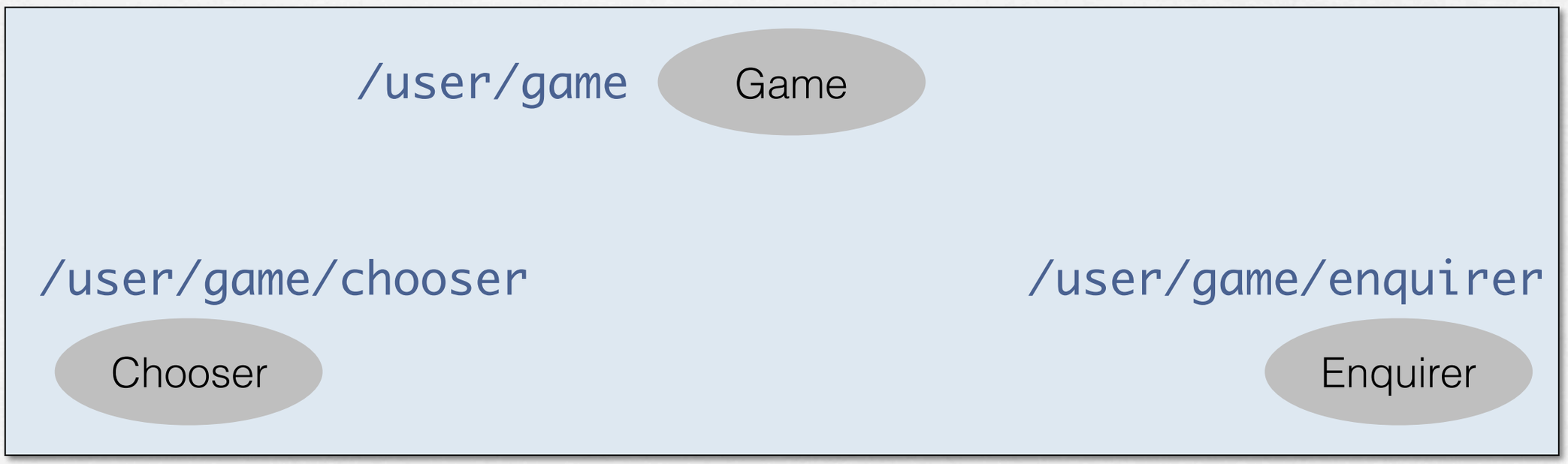
nicht alles selber machen

- [ ] Manifesto: Antwortbereitschaft
- [x] keine blockierenden Aufgaben ausführen
- [x] Kind-Aktoren einsetzen und delegieren
- [x] Fehler im Kind sind folgenlos

- [o] VORSICHT mit `Task<x>` oder `async` sowie `lock`

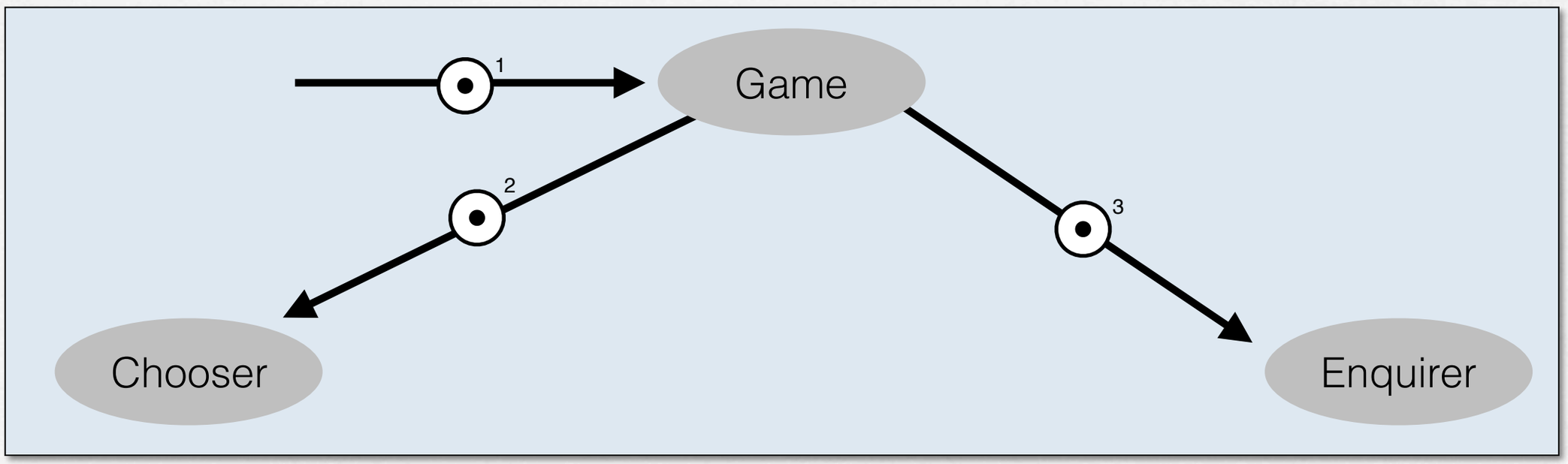
Spiel „Zahlen raten“ (1)

- Ein Aktor denkt sich eine Zahl (1–100) aus
- Ein Aktor versucht die Zahl zu raten
- Ein Aktor koordiniert das Spiel



Spiel „Zahlen raten“ (2)

- Erzeugung
  - Game übernimmt Kontrolle
  - Game legt die beiden Spieler an



Game Actor

```
public class GuessGame : ReceiveActor
{
    private IActorRef chooser;
    private IActorRef enquirer;

    public GuessGame()
    {
        chooser = Context.ActorOf(
            Props.Create<Chooser>(),
            "Chooser"
        );

        enquirer = Context.ActorOf(
            Props.Create<Enquirer>(Self, chooser),
            "Enquirer"
        );
```

andere Aktoren, die wir kennen müssen

Kind-Aktoren erzeugen

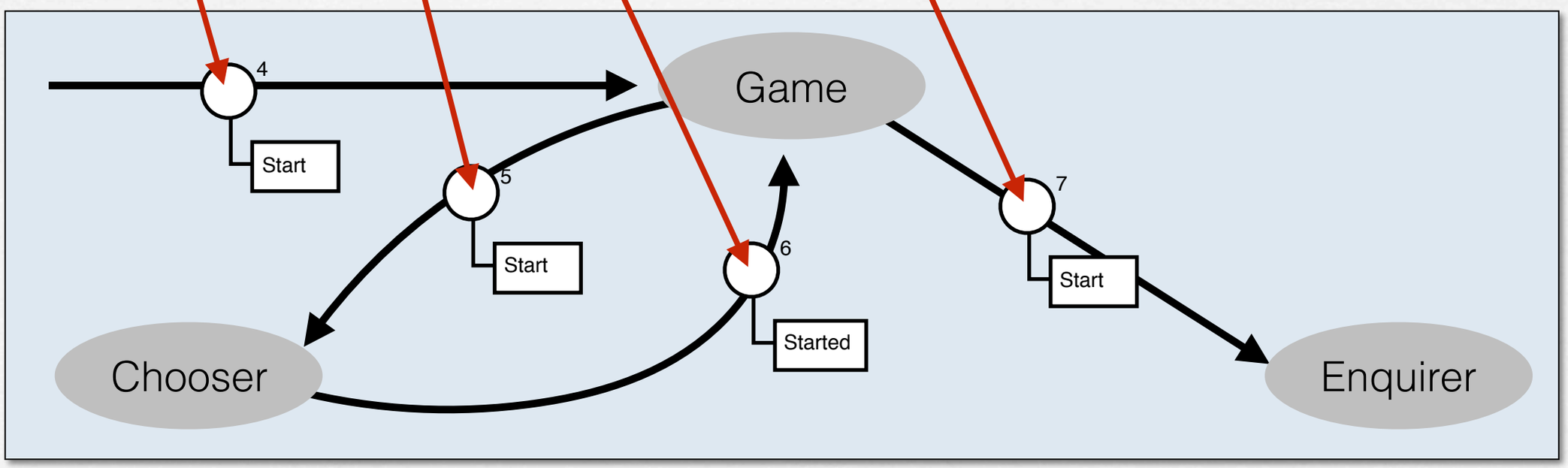
Spiel „Zahlen raten“ (3)

- Initialisierung:
  schön brav nacheinander

```
Receive<Start>(s =>
    chooser.Tell(s)
);
```

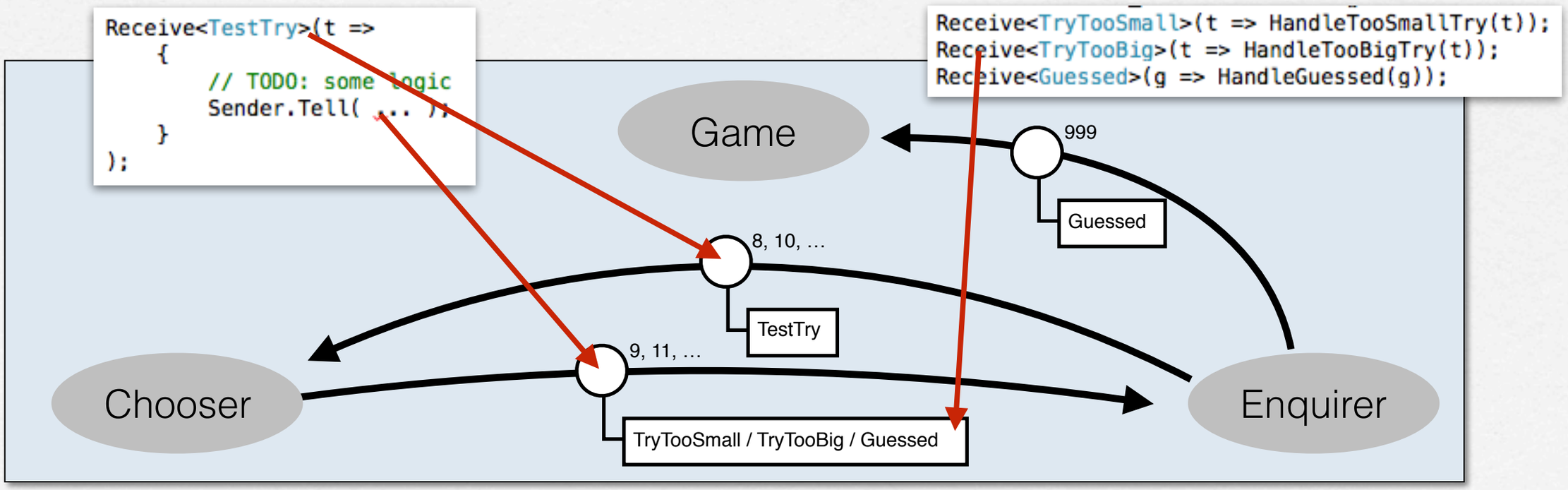
```
Receive<Started>(_ =>
    enquirer.Tell(new Start())
);
```

```
namespace GuessMyNumber.Messages
{
    /// Command to start a new game...
    public class Start
    {
    }
}
```

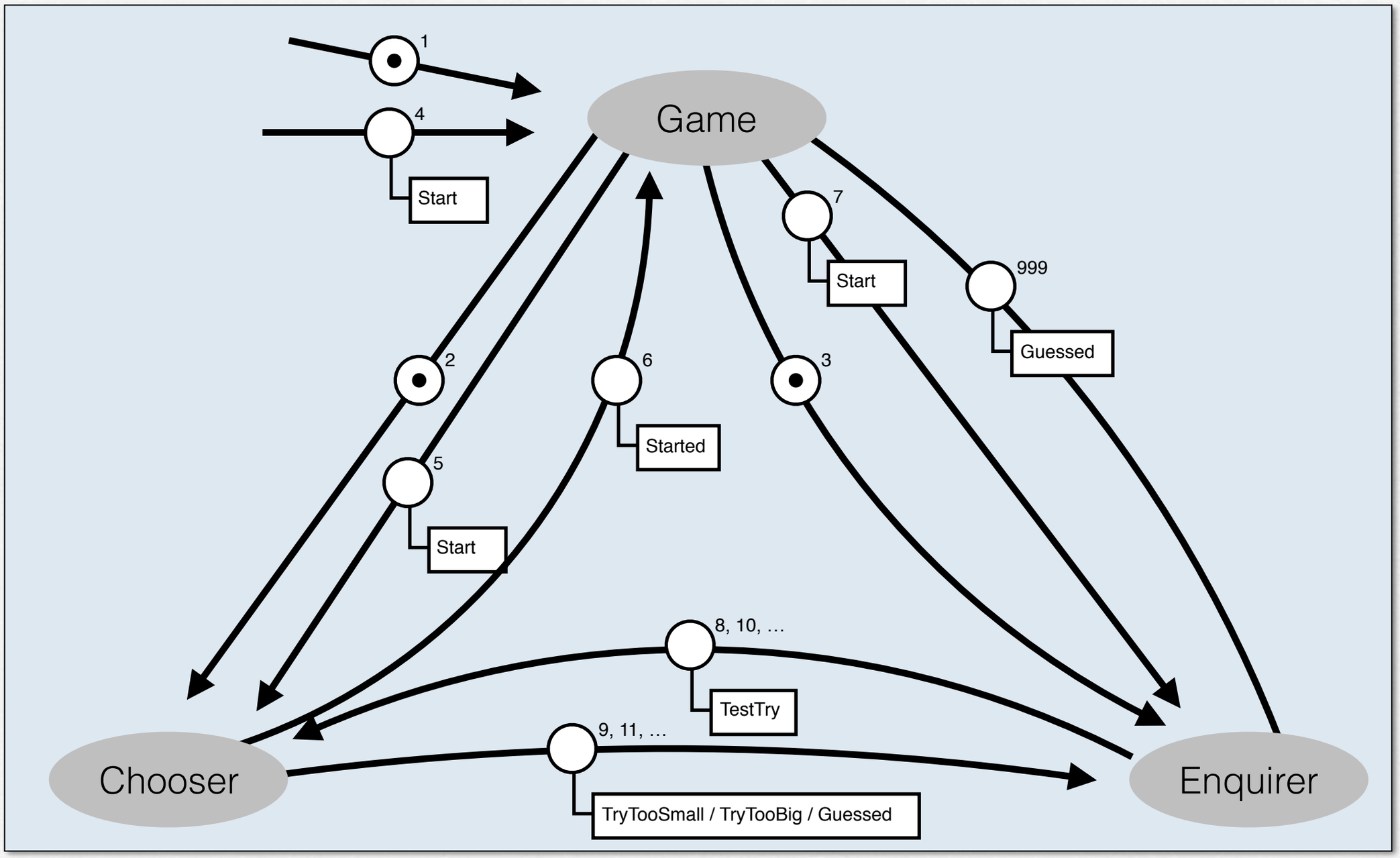


Spiel „Zahlen raten“ (4)

- Der Ratevorgang:
  - Enquirer beginnt
  - danach „ping-pong“



Spiel „Zahlen raten“ (5)



Nachrichten-Typen

- Command: Imperativ

  `Start / TestTry`

- Event: Vergangenheit

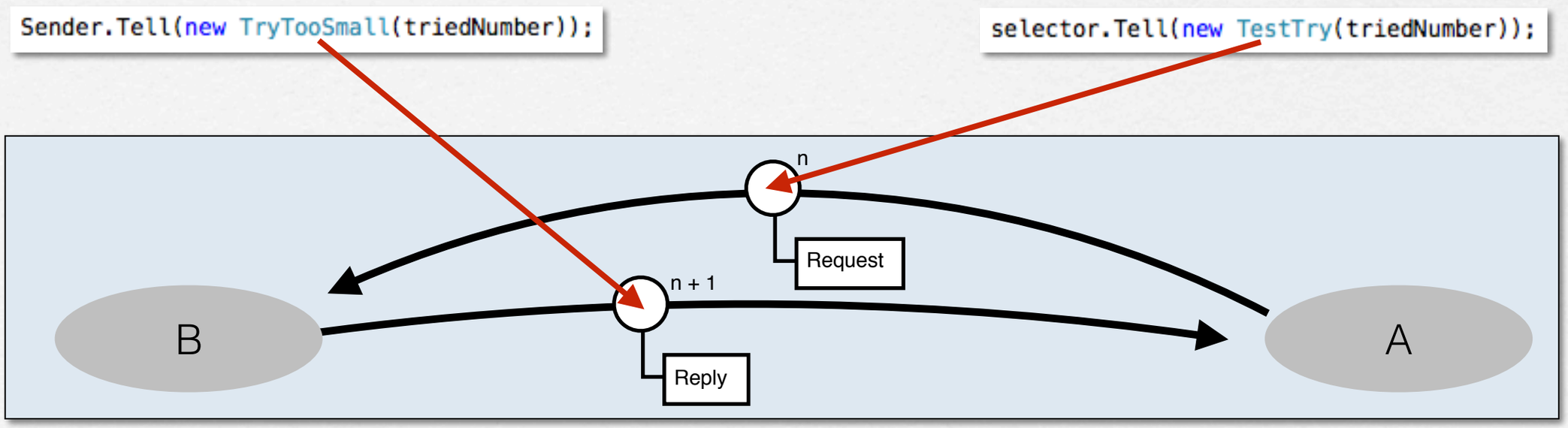
  `Started / Guessed`

- Document: Substantiv

  `TryTooSmall / TryTooBig`

Pattern: Request-Reply

- „Anforderung und Antwort“
  - Anforderung an bekannten Autor
  - Antwort an Sender



Was wir falsch machen

- ☐ Console ist eine „Resource“
- ☐ Was passiert, wenn mehrere Aktoren gleichzeitig etwas ausgeben wollen?
- ☑ Lösung: eigener Aktor :-)

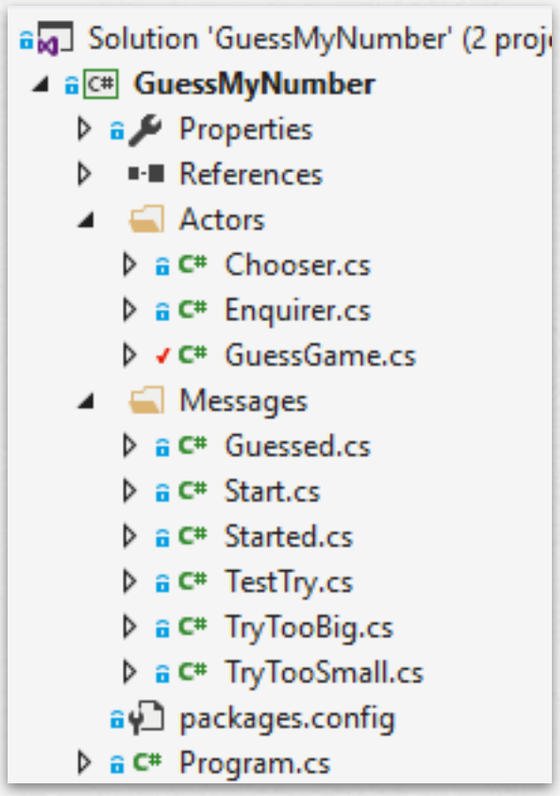
Text2
Text Text2 t1

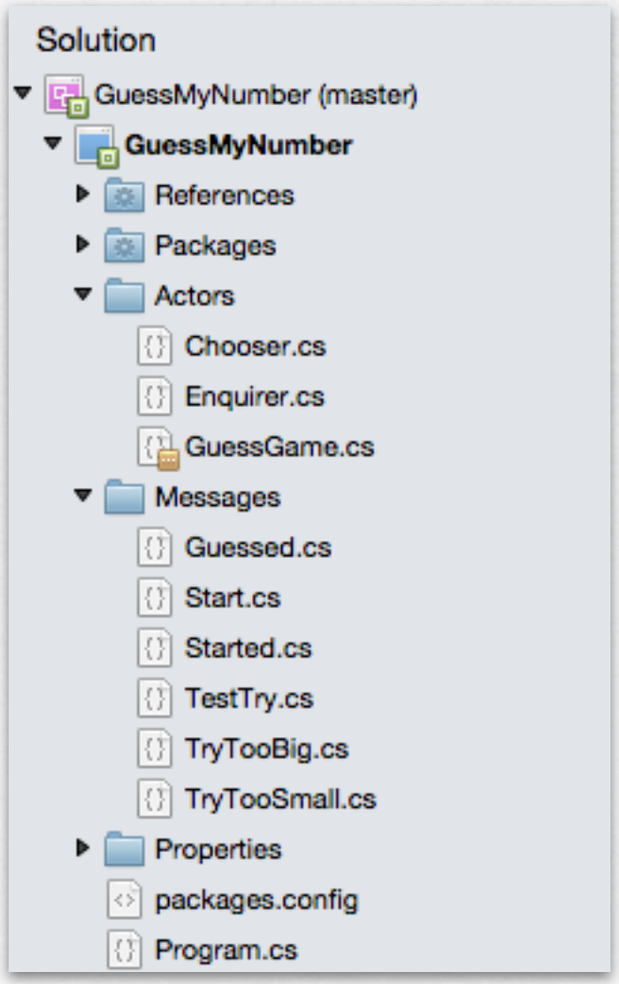
Coding Time

- Solution „GuessMyNumber“
- namespaces für:
  Actors/Messages
- Alternativ: nested

```
namespace PiSequential
{
    public class Master : ReceiveActor
    {
        public class CalculatePi {}
        public Master() { ... }
```

```
master.Tell(new Master.CalculatePi());
```





Aufgabe

```
public class Chooser : ReceiveActor
{
    public int mySecretNumber;

    // ...

    private void HandleTestTry(TestTry testTry)
    {
        var triedNumber = testTry.Number;

        Console.WriteLine("Chooser: Received Guess {0}", triedNumber);

        // TODO: an dieser Stelle müssen wir 3 Fälle behandeln:
        // triedNumber zu klein: TryTooSmall melden
        // triedNumber zu groß: TryTooBig melden
        // triedNumber passt: Guesed melden
        //
        // Nachrichten werden als Antwort versandt nach diesem Muster:
        // Sender.Tell(new MessageClass(args));
    }
}
```

Lösungsvorschlag

```
private void HandleTestTry(TestTry testTry)
{
    var triedNumber = testTry.Number;

    Console.WriteLine("Chooser: Received Guess {0}", triedNumber);

    if (triedNumber < mySecretNumber)
    {
        Sender.Tell(new TryTooSmall(triedNumber));
    }
    else if (triedNumber > mySecretNumber)
    {
        Sender.Tell(new TryTooBig(triedNumber));
    }
    else
    {
        Sender.Tell(new Guessed(triedNumber));
    }
}
```