

# Wolf and Hare

Nils Kohl, Thomas Stadelmayer, David Uhl

Friedrich-Alexander Universität Erlangen-Nürnberg

03.07.2015

## 1 Wolf and Hare

- Spielregeln
- Visualisierung

## 2 Implementierung

- Seriell
- Parallel

- 2D Spielfeld mit zwei Wölfen und einem Hasen
- Zufällige Startposition auf Spielfeld
- Pro Zug: Wölfe jeweils einen Schritt, Hase einen Schritt
- Spielende: Wolf fängt Hasen oder Hase erreicht Ziel

## Aufgabe

- Spiel parallelisieren
- Jede Maschine auf Cluster testet verschiedene Routen

# Visualisierung (1)

x-Position	y-Position
0	0
0	1
0	2
0	3
0	4
1	4
2	4
3	4
4	4

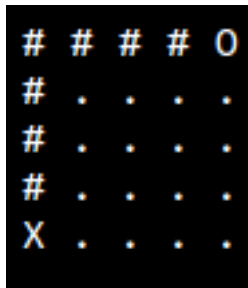
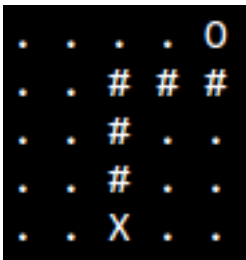
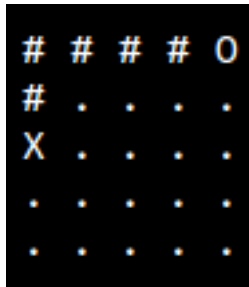


Table : Wolf1

# Visualisierung (2)



RouteX Wolf1



Route Hase

# Visualisierung (3)

x-Position	y-Position
0	0
1	0
2	0
3	0
4	0
4	1
4	2
4	3
4	4

Table : RouteX von Wolf1

x-Position	y-Position
0	2
0	3
0	4
1	4
2	4
3	4
4	4
4	4
4	4

Table : RouteX von Hase

- Routenerstellung für die Wölfe und den Hasen
- Kombination der Wolfsrouten zu Routenpaaren
- Unabhängiger Task: Vergleich eines Routenpaares mit allen Routen des Hasens
- Speicherung der Erfolge bzw. Misserfolge des Routenpaares für alle Hasenrouten
- Vergleich aller Routenpaare

```
for (w1, w2) in wolf_routes do  
  for h in hare_routes do  
    /* test if a wolf catches the hare before it arrives at the special  
    square */ compare (w1, w2) with h;  
    ▷ compares each element;  
    ▷ counter++;  
  
    if caught then  
      | append number of steps needed to list;  
    else  
      | add flag to list (e.g. -1);  
    end  
  end  
end
```



- Tupel von einer Route Wolf1 und Wolf2
- Vergleiche Tupel mit allen Routen von Hase
- Merke Anzahl der Schritte und Wahrscheinlichkeit als Indikatoren
- Wenn Tupel eine bessere Lösung ist, dann lösche alle alten Werte und schreibe Tupel in Liste
- Tupel auf Prozessoren aufteilen
- Nach Berechnung vergleiche die Listen

```
<r1w1,r2w2> = choose tuple of routes wolf1 and wolf2;  
for all hare routes do  
    | compare tuple with route_i of hare;  
        | ▶ compares each element;  
        | ▶ in case of success probability_local++;  
end
```

.  
. .  
.

```
if probability_local < probability_global then  
    probability_global = probability_local;  
    delete all elements in list;  
    add <r1w1, r2w2> to list;  
end  
if p_local == p_global AND counter_local < counter_global then  
    counter_global = counter_local;  
    delete all elements in list;  
    add <r1w1, r2w2> to list;  
end  
if p_local == p_global AND counter_local == counter_global then  
    add <r1w1, r2w2> to list;  
end
```

**Algorithm 1:** How to get best routes

- Sammle alle `p_global` und `counter_global` (MPI-Allgather)
- Root: entscheide welcher Rank die besten Werte hat
- Teile jedem Rank mit ob er die besten hat (1) oder nicht (0) (MPI-Broadcast)
- Jeder Rank, der eine 1 erhalten hat, sendet seine ermittelten Routen zu root (MPI-Send)