Multi-player Strategies for Drafting Games with Applications to Risk

By: Neesha Desai

Richard Gibson

Richard Zhao

The Board Game Risk

Image from wikipedia.org

The Computer Game Lux

Screenshot of Lux Delux

The Computer Game Lux

- **Drafting**
- **Post-Draft Play**

Outline

- Motivation and Problem
- Related Work
- Our Approaches
- Reward Signal
- Results
- Strengths/Weaknesses
- Applications to Video Games
- Future Work

Motivation and Problem

- Why is Drafting problem interesting?
  - Adversarial
  - Multi-player
  - Generalization of a variety of problems
- Not zero-sum when more than 2 players
  - Cannot apply minimax search

Drafting Games

- Finite, deterministic, full information
- n ≥ 2 players
- Game consists of a set of picks or actions
  - Not replaced
- Players take turn taking picks, one at a time
- Game ends when no more picks are allowed by game rules

Drafting Games

- At the end of the game, the set of picks are partitioned into {A1, A2, … An, A}
  - A1, A2, …,An are picks by players 1,2,…n.
  - A is the set of remaining picks, if any
- A reward signal is calculated based on this partition for each player.
- Each player's goal is to make picks as to maximize their own the reward signal

Related Work

- **Minimax Adversarial Search**
  - **Two players zero-sum game**
  - **Game tree search**
  - **Maximize the score of my choice, assuming the opponent is minimizing the score of their choice**
  - **Heuristic function to reduce search depth**

Related Work

- **MaxN**
  - **Assuming that all players are trying to maximize their own payoffs**
- **Paranoid**
  - **Assuming that all other players are trying to minimize the active player's payoff**

Related Work

- **Single-agent drafting**
  - **Finding a set of actions such that the result of the actions is as close to optimal as possible in the problem domain**
  - **Heuristic search combined with a machine-learned fitness function**

Our Approaches

- **Reinforcement Learning**
  - **Sarsa(lambda)**
- **UCT**
- **Kth Best Pick**

NEW!

Reinforcement Learning

- **Sarsa(lambda)**
  - **Set of states**
  - **Set of actions**
  - **At each step, take action (according to a policy, e.g. epsilon-greedy) based on Q(s,a)**
  - **Receive a reward**
  - **Update the Q(s,a) function according to reward**

Sarsa(lambda) applied to Risk

- **Action abstraction**
  - **choose the most empty continent;**
  - **choose the least empty continent;**
  - **choose the continent with the most number of my territories;**
  - **choose the continent with the least number of my territories;**
  - **choose the smallest available continent;**
  - **choose the largest available continent;**
  - **choose the continent with the most access points (links to other continents);**
  - **choose the continent with the least access points.**

Sarsa(lambda) applied to Risk

- State abstraction
  - I am the sole owner of a continent;
  - I have more than half of all territories in a continent;
  - an opponent has more than half of all territories in a continent;
  - there is an empty continent.
- action resolution mechanism is invoked to pick a territory within the chosen continent

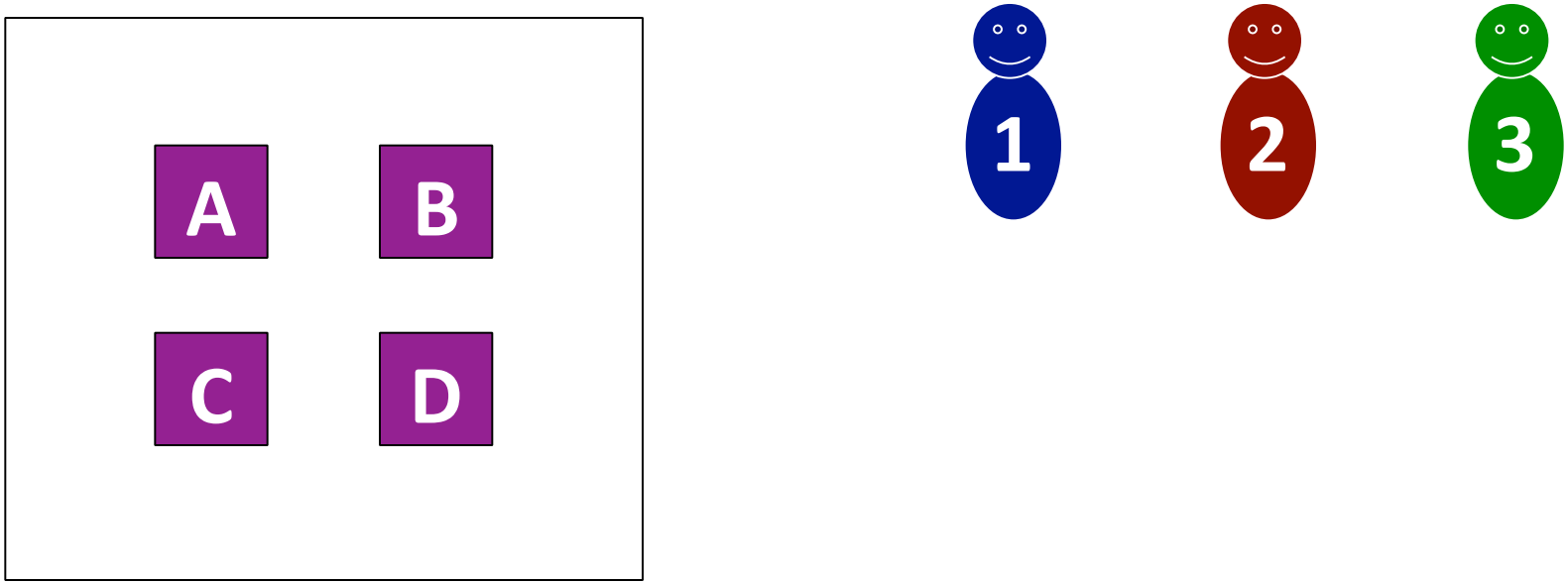
UCT

- **Monte Carlo tree search**
- **Each decision node *s* of player *i* stores *Q*(*s, a*) = average of $r_i(Z)$ seen on simulations where *a* was taken at *s*.**
- **During simulations, nodes expanded by taking**

$$a = \arg\max_{a'} Q(s,a') + c\sqrt{\frac{\log(n(s))}{n(s,a')}}$$

- **After simulations, UCT takes action** $a = \arg\max_{a'} Q(s,a')$

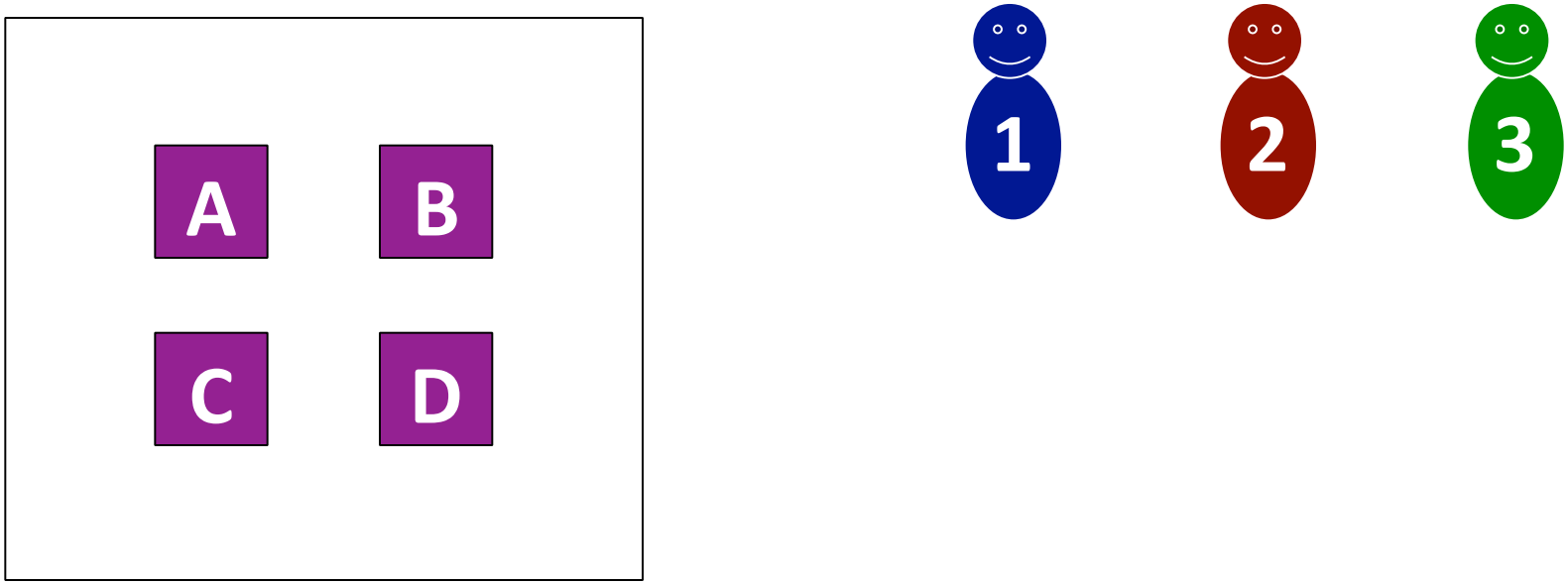
An Example Drafting Game



4 + 1 = 5

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

An Example Drafting Game



3 + 4 = 7

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

The KthBestPick Algorithm

- Essentially, just an encoding of this reasoning
- Specifically for drafting games only
- Uses a heuristic to rank actions from best to worst
- Makes assumptions about how opponents will play

KthBestPick Pseudocode

- **Inputs:**
  - **Current state** s
  - **Number of top picks to consider** N
  - **Heuristic** h
  - **Opponent models** $m_1, \ldots, m_n$
- **Output:**
  - **Action to take** a

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' Є betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })

N = 2

rankedPicks = (B, A, C, D)

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })

N = 2

rankedPicks = (B, A, C, D)

k = 1

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })

N = 2

rankedPicks = (B, A, C, D)

k = 1

a = A

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })

N = **2**

rankedPicks = **(B, A, C, D)**

k = **1**

a = **A**

betterPicks = **(B)**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })

N = **2**

rankedPicks = **(B, A, C, D)**

k = **1**

a = **A**

betterPicks = **(B)**

s' = { {A}, { }, { } }

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A}, { }, { } }
makeThisPick = **TRUE**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = **{ {A}, { }, { } }**
makeThisPick = **TRUE**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = 2
rankedPicks = (B, A, C, D)
k = 1
a = A
betterPicks = (B)
s' = { {A}, { }, { } }
makeThisPick = TRUE
p = 2

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = **{ {A}, { }, { } }**
makeThisPick = **TRUE**
p = **2**
a' = **D**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' Є betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = 2
rankedPicks = (B, A, C, D)
k = 1
a = A
betterPicks = (B)
s' = { {A}, {D}, { } }
makeThisPick = TRUE
p = 2
a' = D

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A}, {D}, { } }
makeThisPick = **TRUE**
p = **2**
a' = **D**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = **{ {A}, {D}, { } }**
makeThisPick = **TRUE**
p = **2**
a' = **D**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A}, {D}, { } }
makeThisPick = **TRUE**
p = **3**
a' = **D**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = **{ {A}, {D}, { } }**
makeThisPick = **TRUE**
p = **3**
a' = **C**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = 2
rankedPicks = (B, A, C, D)
k = 1
a = A
betterPicks = (B)
s' = { {A}, {D}, {C} }
makeThisPick = TRUE
p = 3
a' = C

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A}, {D}, {C} }
makeThisPick = **TRUE**
p = **3**
a' = **C**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = **{ {A}, {D}, {C} }**
makeThisPick = **TRUE**
p = **3**
a' = **C**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A}, {D}, {C} }
makeThisPick = **TRUE**
p = **1**
a' = **C**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A}, {D}, {C} }
makeThisPick = **TRUE**
p = **1**
a' = **B**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' Є betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = 2
rankedPicks = (B, A, C, D)
k = 1
a = A
betterPicks = (B)
s' = { {A, B}, {D}, {C} }
makeThisPick = TRUE
p = 1
a' = B

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A, B}, {D}, {C} }
makeThisPick = **TRUE**
p = **1**
a' = **B**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = **2**
rankedPicks = **(B, A, C, D)**
k = **1**
a = **A**
betterPicks = **(B)**
s' = { {A, B}, {D}, {C} }
makeThisPick = **TRUE**
p = **1**
a' = **B**

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' Є betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = 2
rankedPicks = (B, A, C, D)
k = 1
a = A
betterPicks = ( )
s' = { {A, B}, {D}, {C} }
makeThisPick = TRUE
p = 1
a' = B

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

s = ({ } , { }, { })
N = 2
rankedPicks = (B, A, C, D)
k = 1
a = A
betterPicks = ( )
s' = { {A, B}, {D}, {C} }
makeThisPick = TRUE
p = 1
a' = B

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' ∈ betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

	A	B	C	D
Player 1	3	4	2	1
Player 2	4	1	2	3
Player 3	4	2	3	1

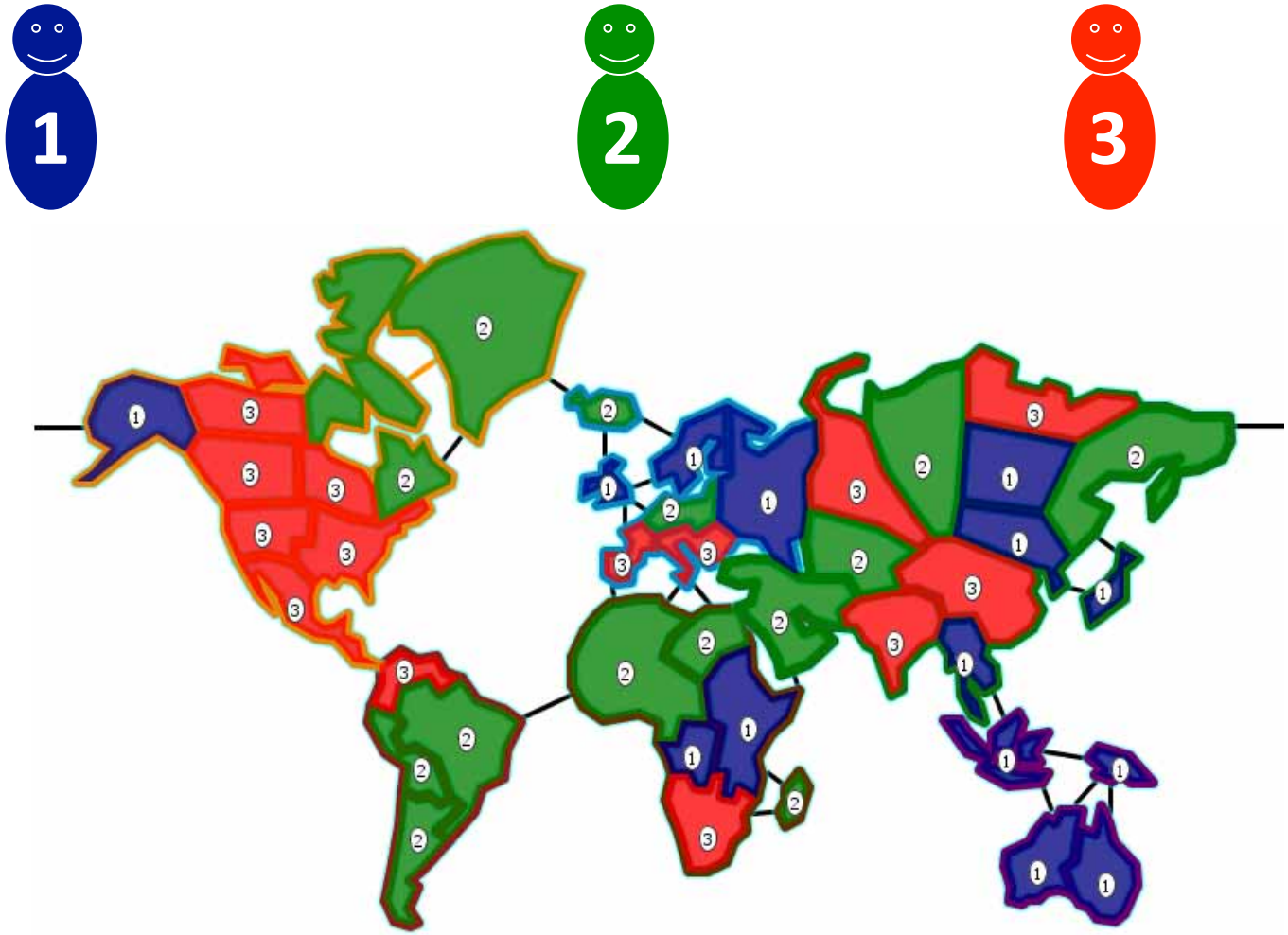
s = ({ } , { }, { })
N = 2
rankedPicks = (B, A, C, D)
k = 1
a = A
betterPicks = ( )
s' = { {A, B}, {D}, {C} }
makeThisPick = TRUE
p = 1
a' = B

Theoretical Properties

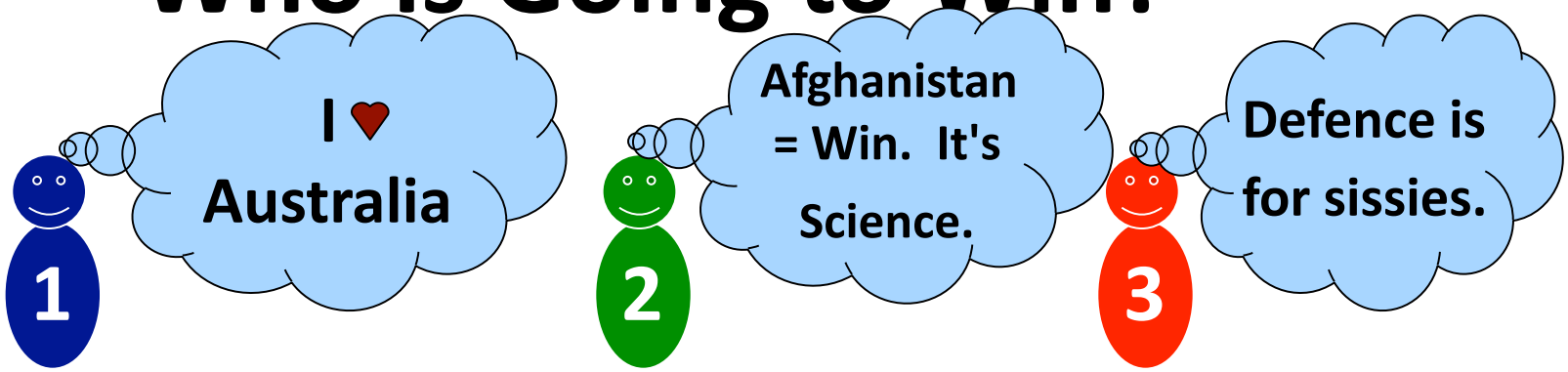
- KthBestPick always returns an action.
  - While loop is not infinite because drafting games are acyclic
  - The for loop will always return an action on its last time through

```
rankedPicks ← sort(s.getActions(), h)
for k from N-1 to 0 do
    a ← rankedPicks(k)
    betterPicks ← rankedPicks(0..k-1)
    s' ← s.nextState(a)
    makeThisPick ← TRUE
    while betterPicks is not empty do
        p ← s'.getActivePlayer()
        a' ← m_p(s')
        s' ← s'.nextState(a')
        if a' Є betterPicks then
            if p = s.getActivePlayer() then
                betterPicks.remove(a')
            else
                makeThisPick ← FALSE and BREAK from while
    if makeThisPick then RETURN a
```

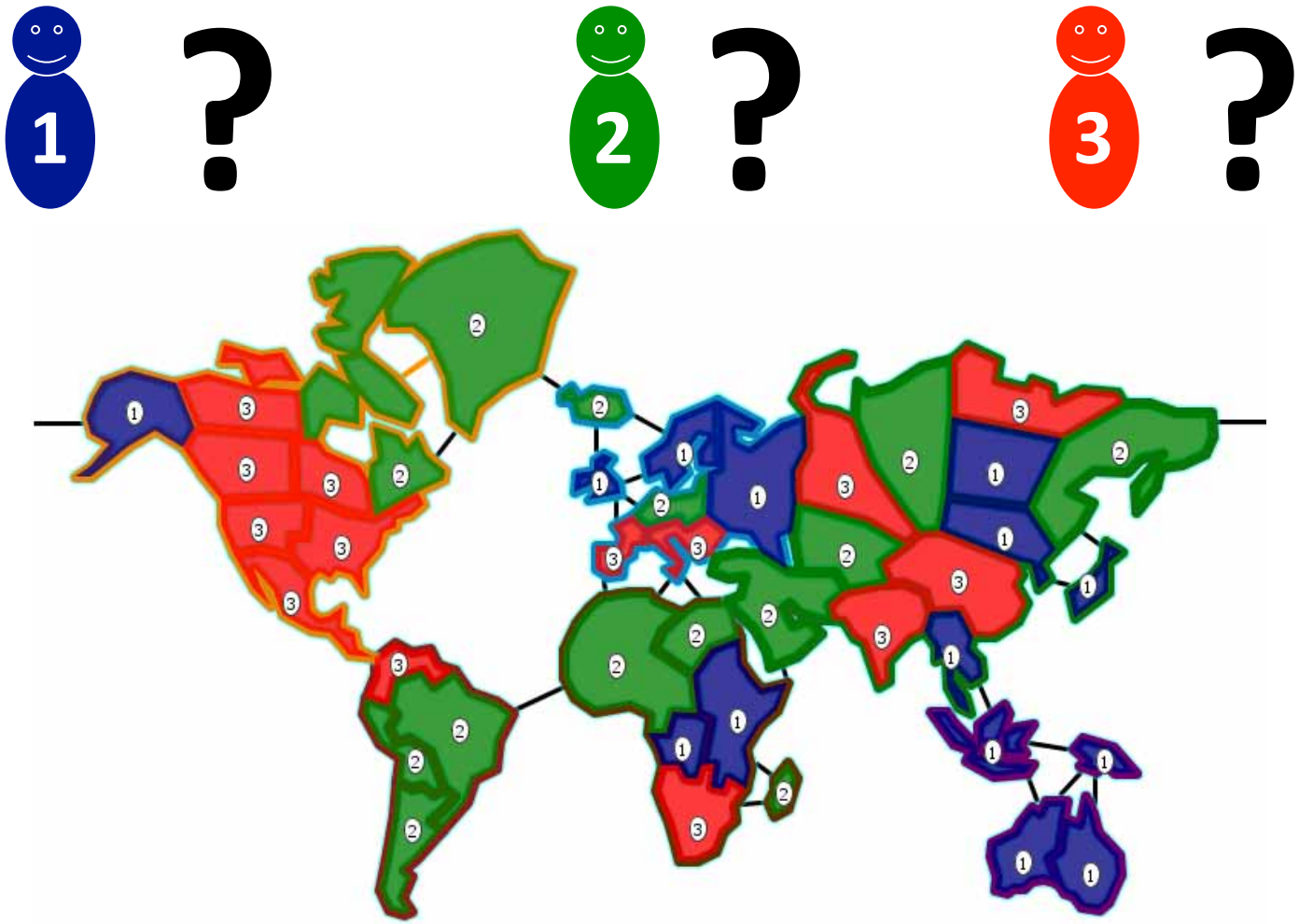
Who is Going to Win?



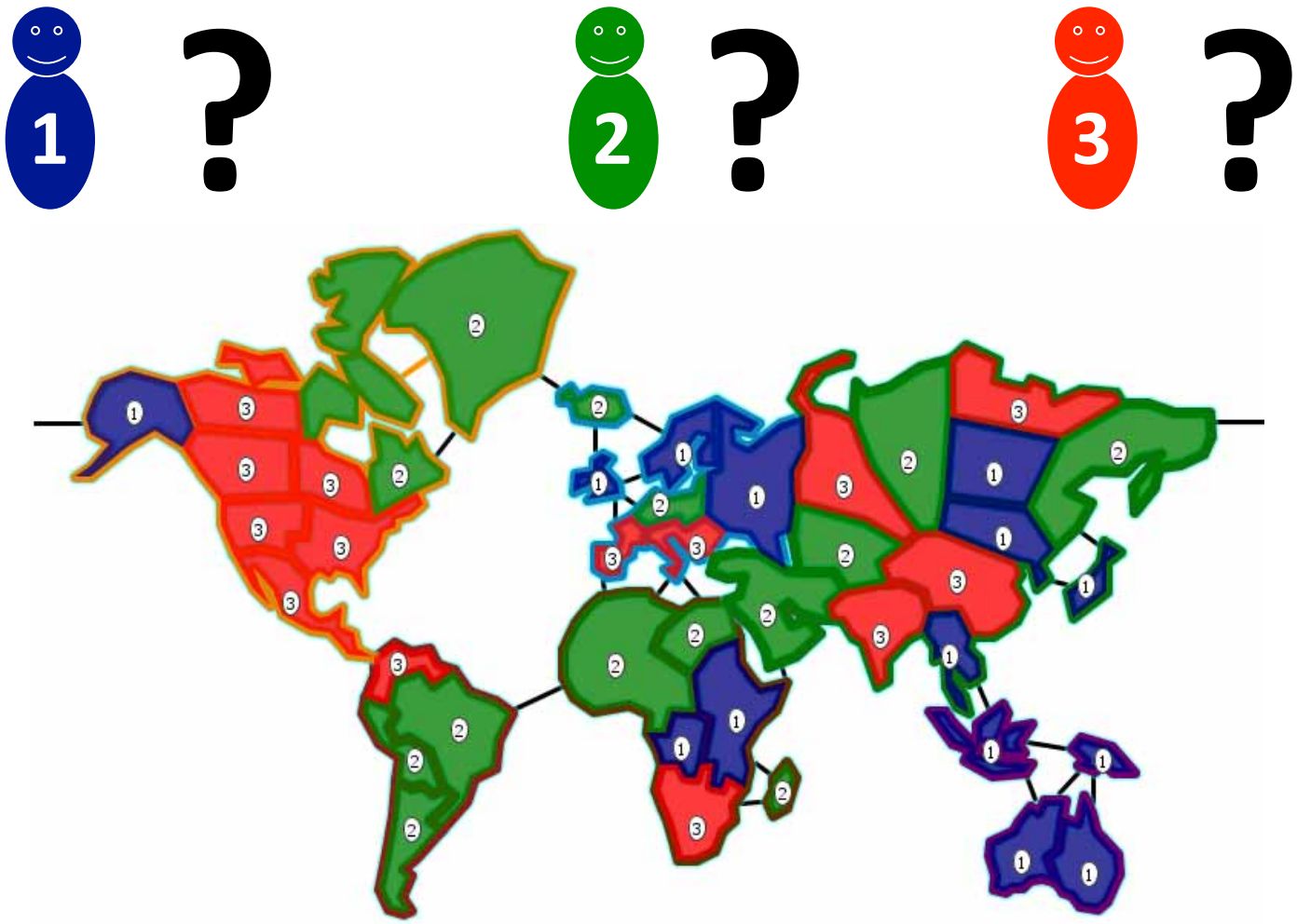
Who is Going to Win?



Who is Going to Win?



How Likely is Each Player To Win?



Engineering a Reward Signal

- **Recall:** We want to find a winning drafting strategy for Risk.

- Let's fix our post-draft strategy: Use “Quo” bot from Lux Delux.

- We want to estimate our chance of winning from any draft outcome.

Engineering a Reward Signal: Step 1

- Identify some important features of Risk draft outcomes:
  - For each continent, the number of territories we own in that continent
  - When we play in the turn order
  - The number of distinct enemy territories neighbouring our own
  - The number of (ordered) pairs of our territories which are adjacent

Feature Set Example

S_1 = (4, 0, 2, 1, 3, 4, 1, 18, 22)

Engineering a Reward Signal: Step 2

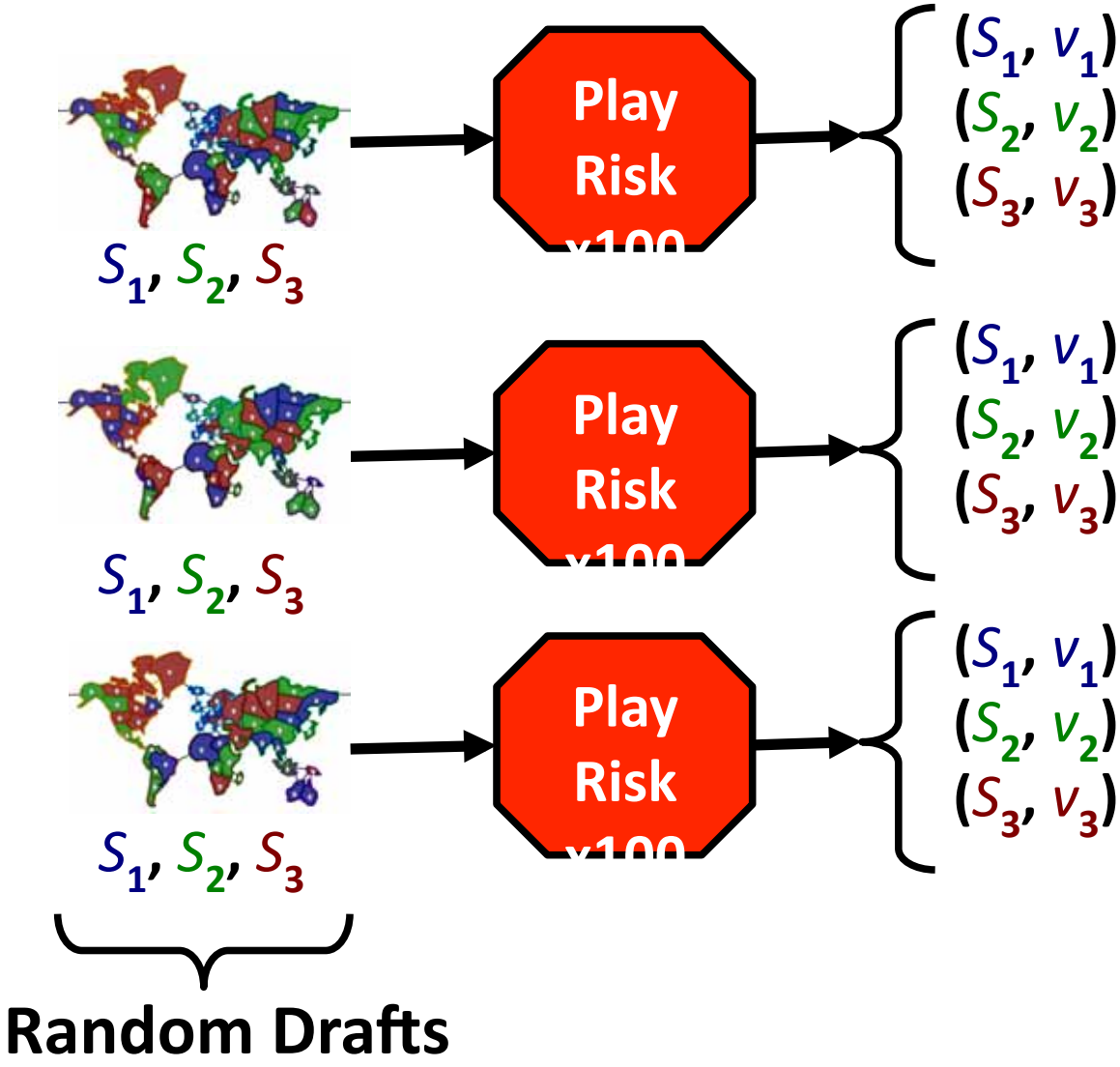
S_1, S_2, S_3

S_1, S_2, S_3

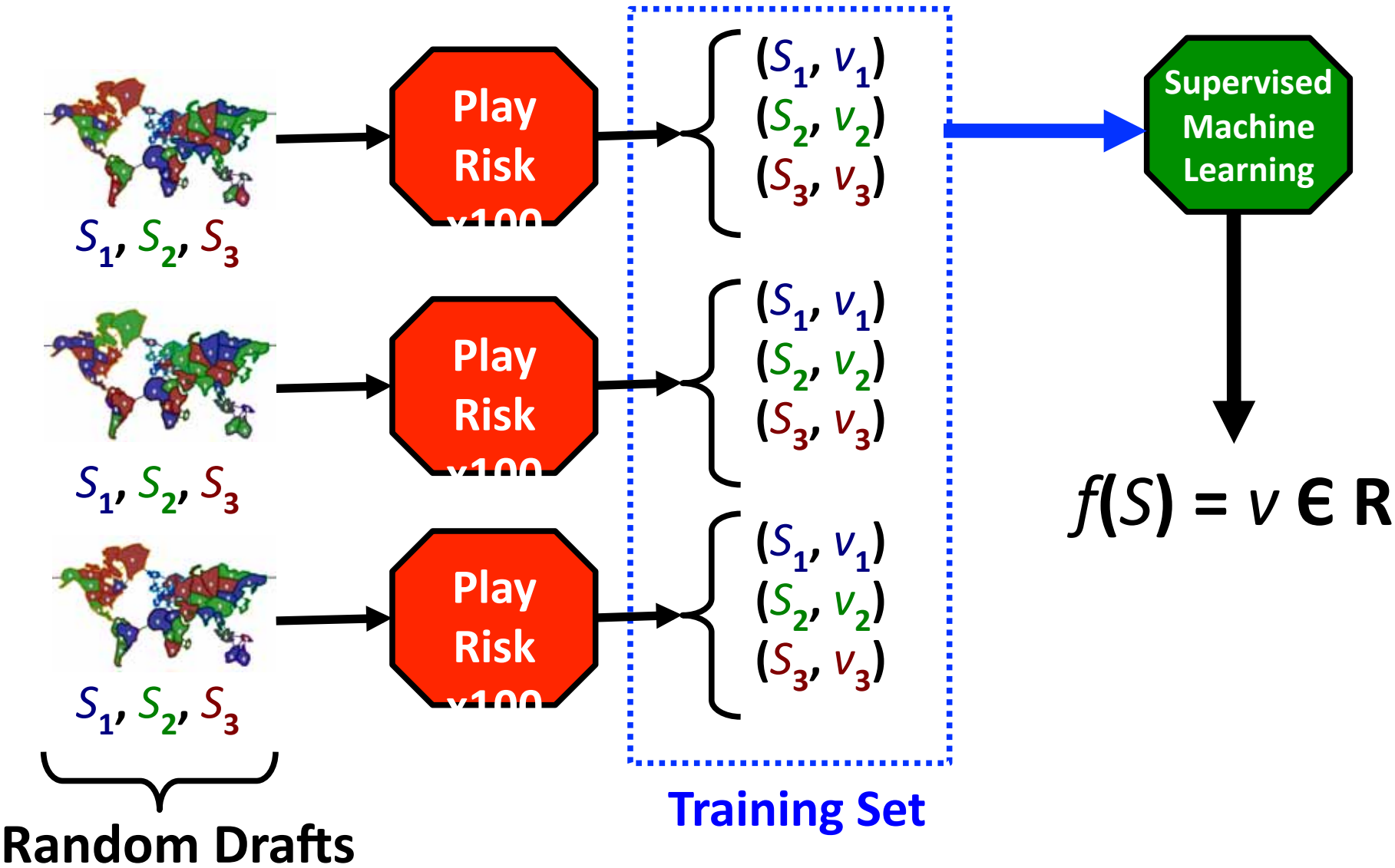
S_1, S_2, S_3

Random Drafts

Engineering a Reward Signal: Step 3



Engineering a Reward Signal: Step 4



Engineering a Reward Signal: Step 5

- $r_i($  $) = \dfrac{f^+(S_i)}{f^+(S_1) + f^+(S_2) + f^+(S_3)}$

where $f^+(S_i) = \max(0, f(S_i))$

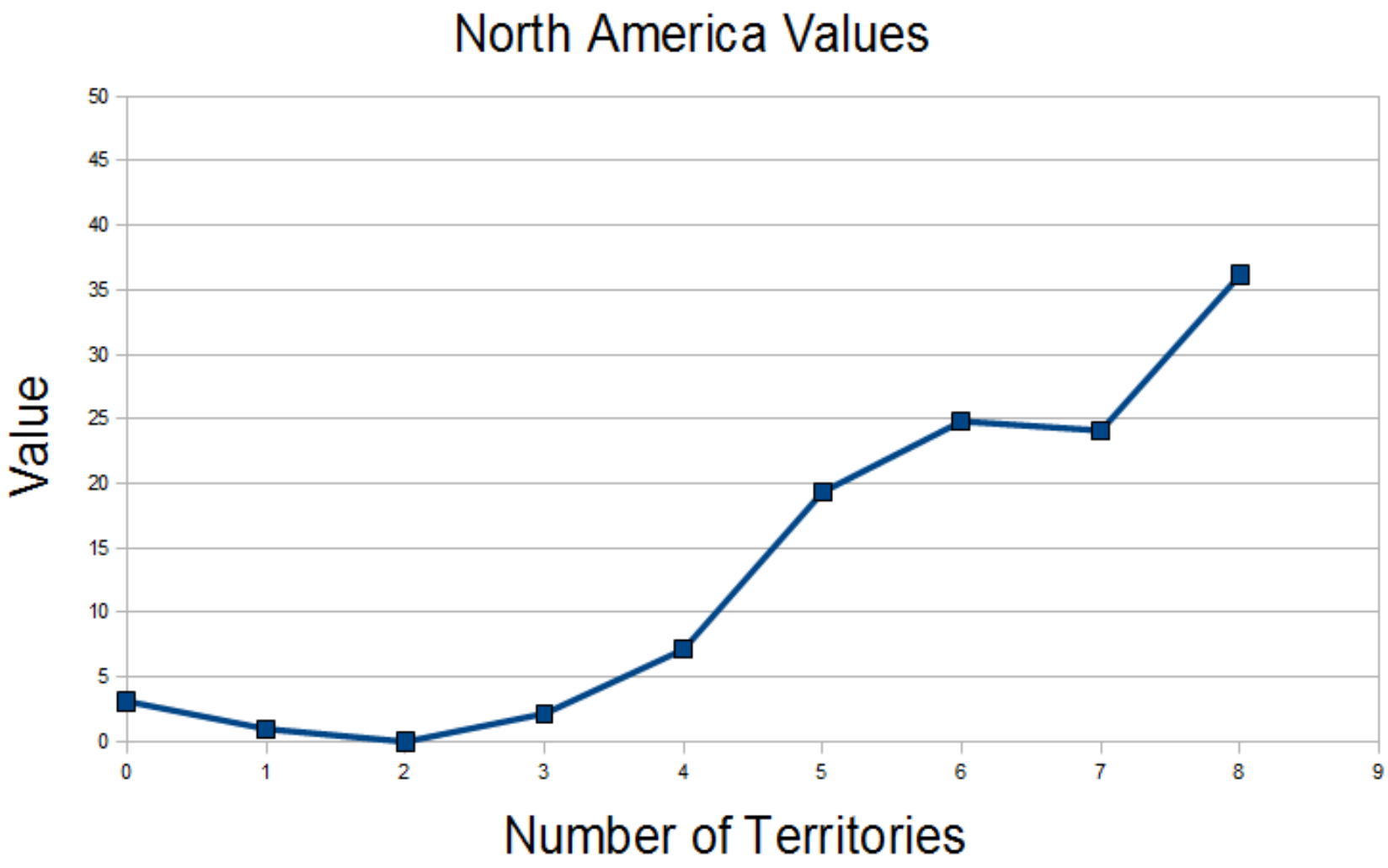
Our Reward Signal

- **Used 7,394 random draft outcomes**
- **Thrown into Weka, used linear regression classifier to obtain** f

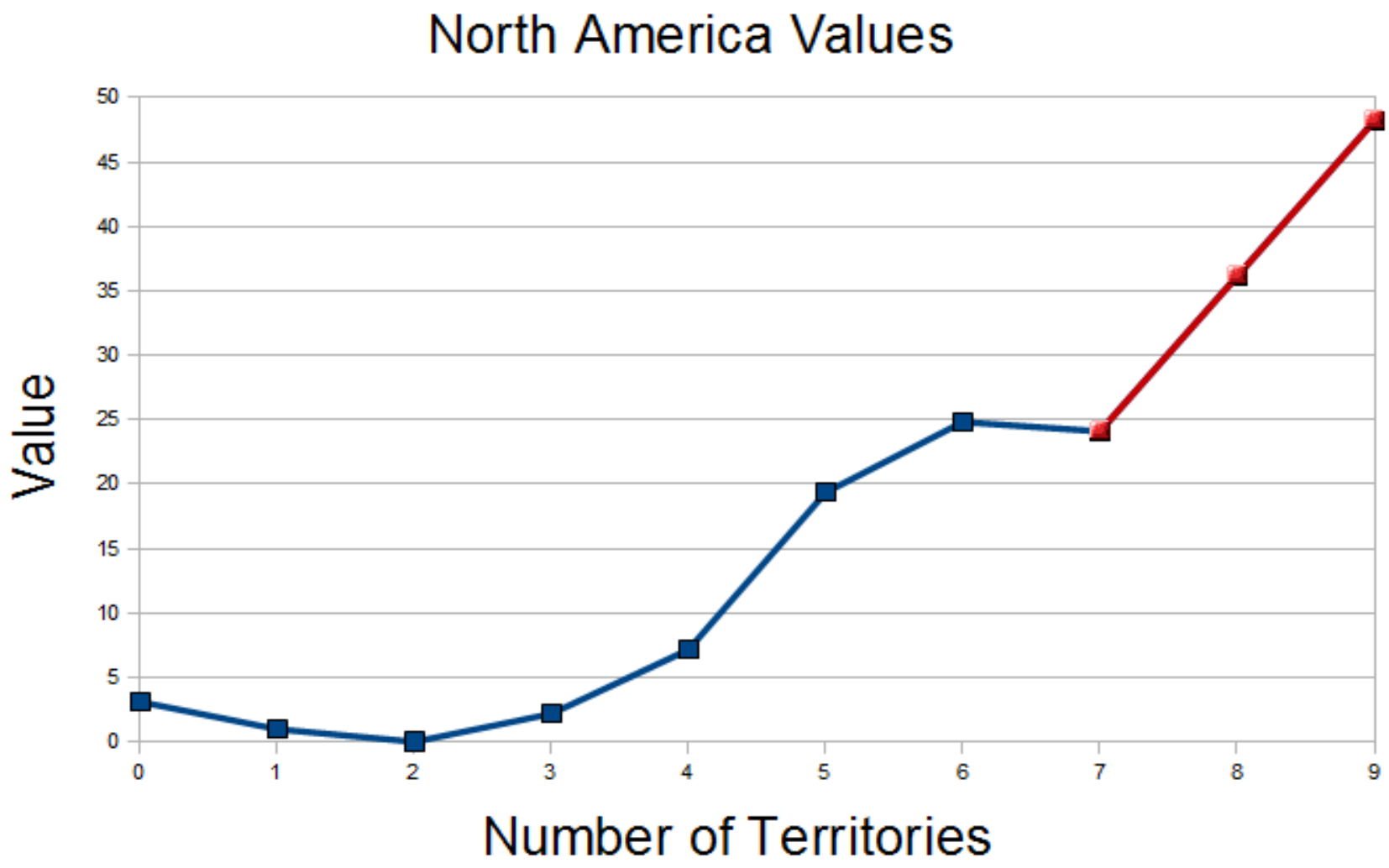
	Australia	South Amer	Africa	North Amer	Europe	Asia
0	2.97	0.69	14.40	3.11	42.44	27.10
1	0	1.23	12.87	0.98	45.11	23.90
2	8.45	3.90	10.72	0	43.11	23.61
3	9.99	0	7.16	2.17	43.77	23.10
4	10.71	17.72	1.23	7.15	41.35	23.61
5	-	-	0	19.35	50.77	23.68
6	-	-	29.80	24.82	43.85	19.32
7	-	-	-	24.10	36.93*	15.63
8	-	-	-	36.15	-	17.43
9	-	-	-	48.20*	-	13.84
10	-	-	-	-	-	10.25*
11	-					6.66*
12	-					3.07*

First to play	13.38
Second to play	5.35
Enemy neighbours	-0.07
Friendly neighbours	0.48

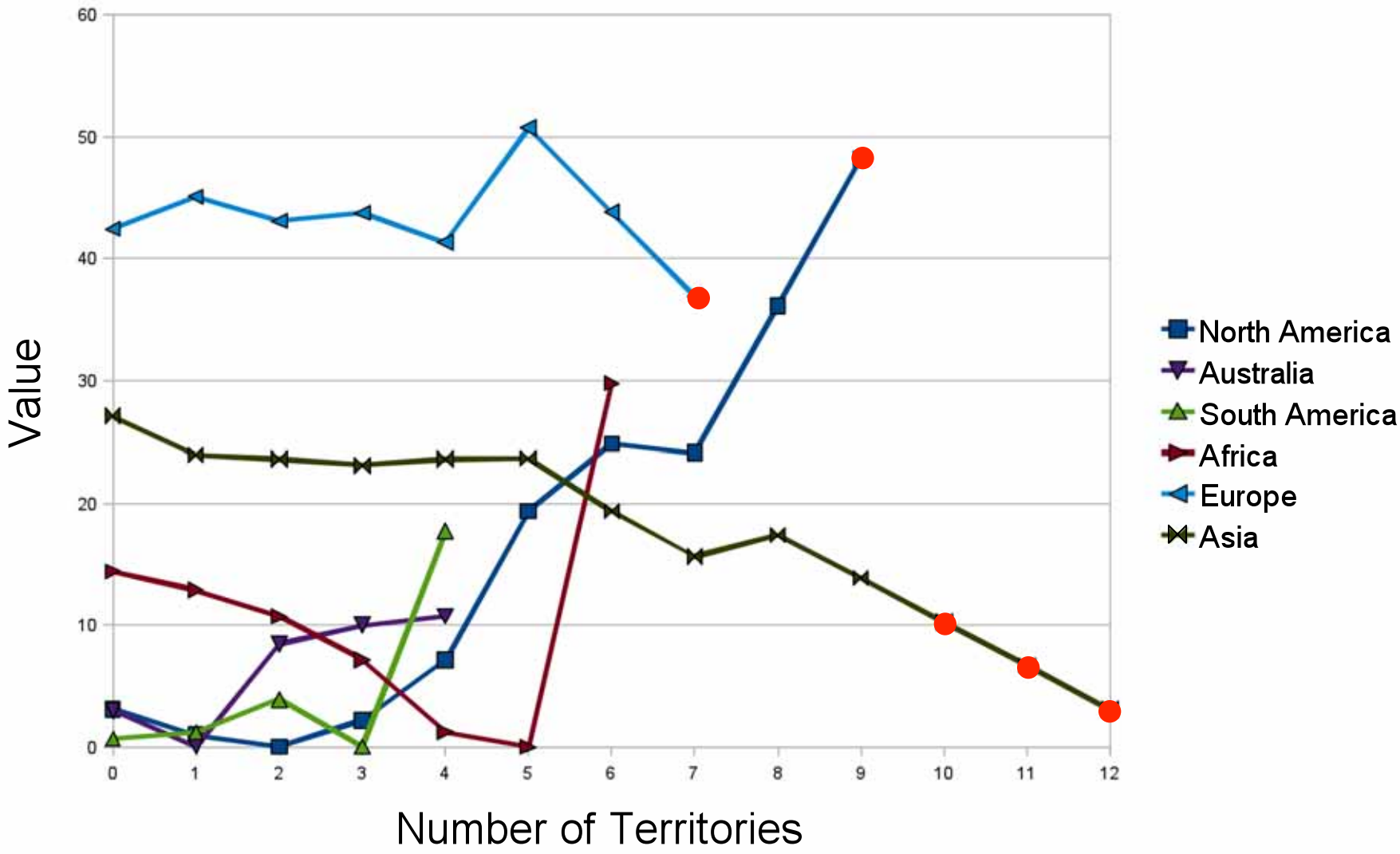
Example of Table Values



Example of Table Values



Continent Table Values



Empirical Evaluation

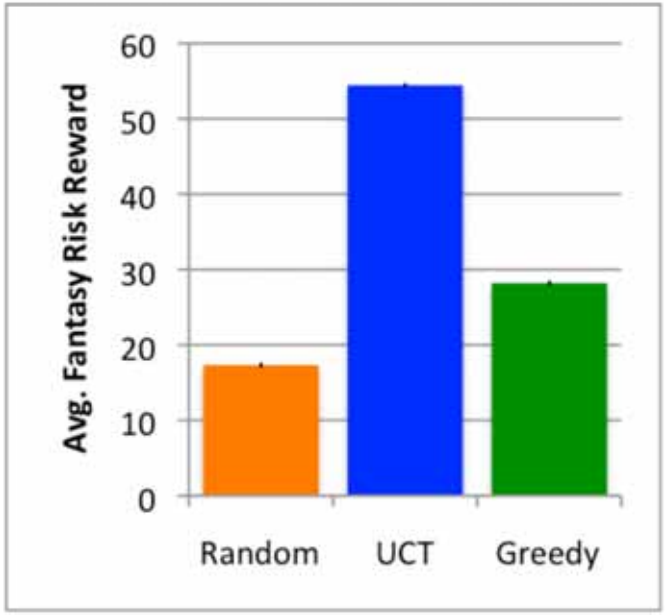
- Two scenarios:
- "Fantasy Risk"
  - Game ends immediately after Risk draft
  - Players score points according to r_i )
- Actual Risk
  - Insert our drafting strategy into "Quo" bot
  - Face-off against difficult bots in Lux Delux

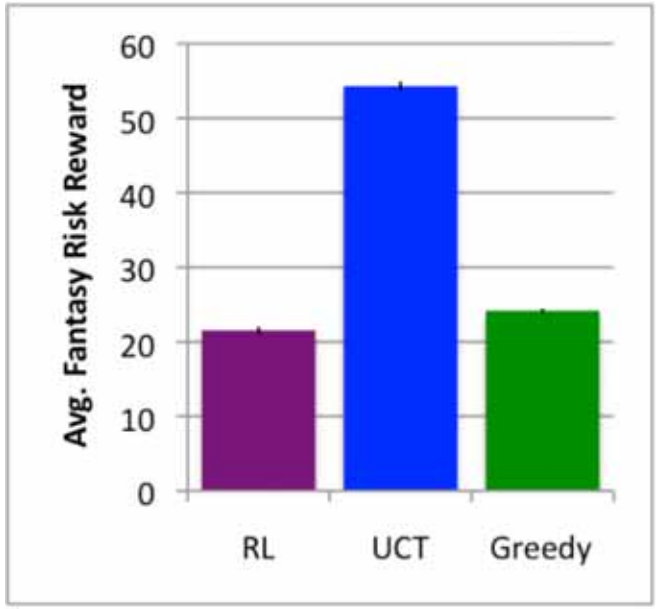
Fantasy Risk

- **Two baseline strategies:**

- **A random drafter**
- **A "greedy" drafter:**
  - **1-ply lookahead**
  - **Evaluate each action using $r_i($  $)$**
  - **Ignore unowned territories**

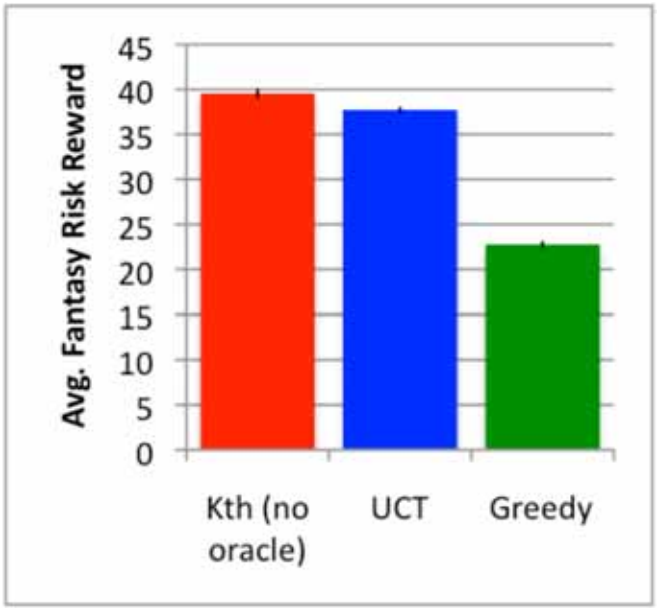
Fantasy Risk Results 1

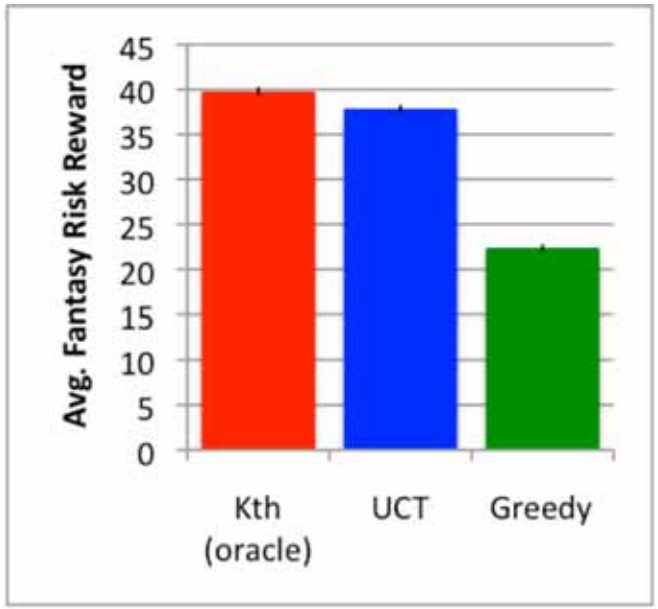




- **Played 100 rounds (1 round = 6 games)**
- **UCT used 3000 simulations, exploration c = 0.01**
- **RL used $\alpha = 0.1$, $\varepsilon = 0.2 / \max(1, t/100)$, $\gamma = 1$, $\lambda = 0.9$**
- **RL initially trained for 100 rounds through self-play**

Fantasy Risk Results 2



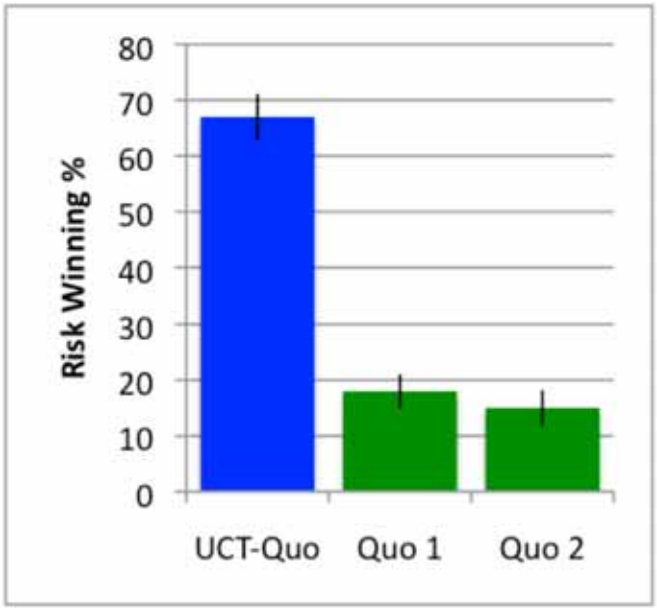


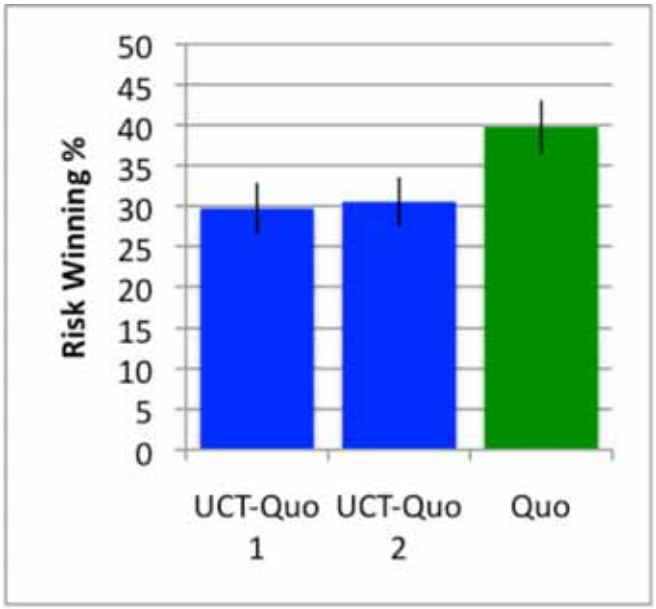
- **Played 50 rounds (1 round = 6 games)**
- **Time limit of 250ms per unowned territory to pick**
- **UCT ran as many simulations as possible, c = 0.01**
- **KthBestPick used UCT(3000,0.25) as heuristic and iterated on N**

Actual Risk

- **Took the top hard bot Quo and replaced it's drafting strategy with Quo**
- **Compared against original Quo and the other 3 hard bots, Evil Pixie, Boscoe and KillBot**

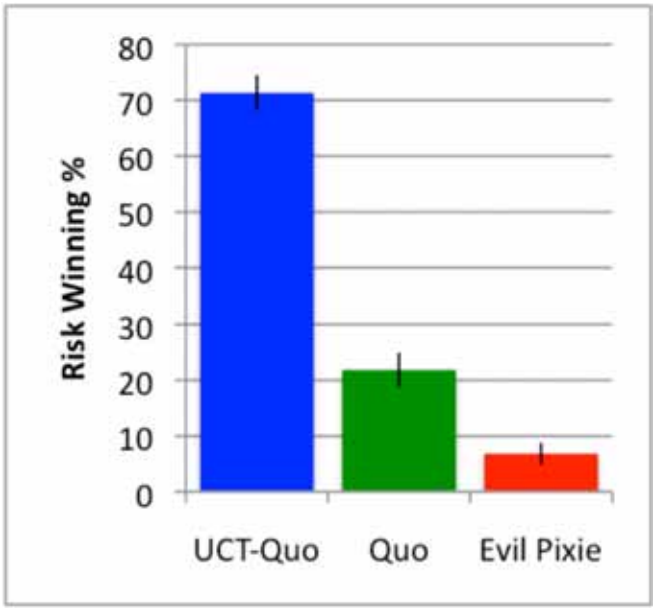
Actual Risk Results 1

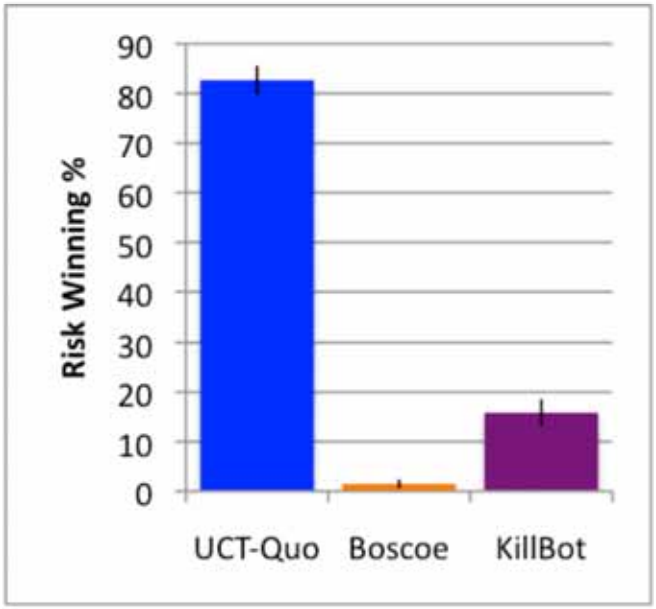




- **Played 100 rounds (1 round = 6 games)**
- **UCT used 3000 simulations, exploration c = 0.01**

Actual Risk Results 2





- **Played 100 rounds (1 round = 6 games)**
- **UCT used 3000 simulations, exploration c = 0.01**

Strengths

- UCT-Quo was able to beat all other “hard” bots
  - Improved post-game by improving draft strategy
- KthBestPick slight improvement over UCT
  - Can we learn opponent model as we play?

Weaknesses

- KthBestPick is too slow versus UCT in Risk
  - Can it be made faster?
- UCT-Quo vs UCT-Quo cancel each other out
  - Nature of UCT or Reward signal?
- Reward signal tied specifically to one map and 3 players
  - Can it be generalized to other games?

Application to Video Games

- Other drafting games:
  - Sports games like Hockey, Baseball, etc
    - KthBestPick more applicable (obvious heuristic)
  - Star Wars: Empire at War
- Difficulty levels
- Other complete drafting games?

Future Work

- Apply our techniques to other drafting games, such as drafting in fantasy sports
- Use a more sophisticated classifier for engineering a reward signal
- Explore better RL abstractions
- Investigate link between draft and post-game play
- General reward signal (eg. work on multiple maps/2-6 players)

Questions?

?