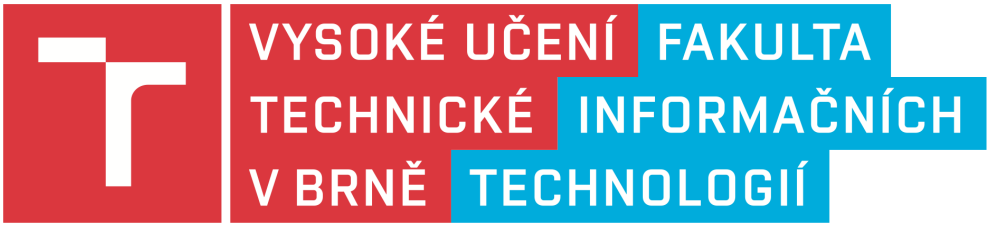

SUI 2020/2021

DICE WARS

Aneta Dufková (xdufko02)
Michal Janů (xjanum03)
Tomáš Kožár (xkozar02)
Martin Mazánek (xmazan09)

1 Baseline – náhodné chování

Jako baseline byla využita náhodná implementace z `rand.py`. Ta byla zkopírována do naší `xdufko02` a v turnajích 4 hráčů si vedla následovně:

```
.        .     % winrate [ .  / .   ] dt.rand  dt.sdc   dt.ste   dt.wpm_c xlogin00 xdufko02
dt.ste   52.59 % winrate [ 61 / 116 ] 56.2/48  46.2/52  52.6/116 56.7/60  51.4/72  52.6/116
dt.wpm_c 41.13 % winrate [ 51 / 124 ] 44.1/68  41.1/56  30.0/60  41.1/124 48.4/64  41.1/124
dt.sdc   31.03 % winrate [ 36 / 116 ] 38.3/60  31.0/116 26.9/52  25.0/56  32.8/64  31.0/116
dt.rand  13.79 % winrate [ 16 / 116 ] 13.8/116 15.0/60  16.7/48  7.4/68   17.9/56  13.8/116
xdufko02 12.00 % winrate [ 24 / 200 ] 12.9/116 14.7/116 7.8/116  11.3/124 13.3/128 12.0/200
xlogin00 9.38  % winrate [ 12 / 128 ] 8.9/56   10.9/64  8.3/72   9.4/64   9.4/128  9.4/128
```

Obrázek 1: Podle očekávání je na tom náhodně hrající `xdufko02` podobně jako `dt.rand`.

2 ExpectiMiniMax – první verze

Na začátku jsme se rozhodli řešit problém tak, že uděláme prohledávání stavového prostoru a pro ohodnocování jednotlivých stavů použijeme natrénovaný model. Prvním krokem tedy bylo prohledávat stavový prostor.

Algoritmus `MaxN` (MiniMax pro více než 2 hráče) velmi pěkně popisují Carol A. Luckhardt a Keki B. Irani [1]. Z jejich paperu jsme si odnesli zejména následující dvě myšlenky:

1. pro více hráčů nestřídáme min a max, nýbrž bereme vždy max
2. stavy máme ohodnoceny ne jedním číslem, ale vektorem o délce n, kde n je počet hráčů (každý prvek vektoru ohodnocuje stav pro daného hráče)

Ve finále jsme však použili jen myšlenku číslo jedna. V naší hře se totiž hráči nestřídají po jednom tahu. Hráč, který je na řadě, může udělat libovolný počet tahů, takže nemůžeme prohledávat stavový prostor a předpokládat, že v každé úrovni se snaží jeden hráč maximalizovat svůj zisk.

Nakonec prohledáváme stavový prostor jen z hlediska hráče na tahu (našeho `xdufko02`), tedy díváme se dopředu pouze na jeho tahy. Díky tomu ani nepotřebujeme vektor pro ohodnocení stavů, stačí nám jedno číslo, které ohodnocuje výhodnost stavu pro hráče `xdufko02`.

Navíc do hry vstupuje ještě pravděpodobnost, že se políčko podaří získat. Na tu využíváme funkci `attack_succcess_probability()`.

2.1 Ohodnocení stavů

Výhledově jsme chtěli nasadit model, pro první implementaci algoritmu `ExpectiMiniMax` jsme použili triviální ohodnocení stavu, a sice hráčovo skóre (počet souvislých polí).

Kupodivu jsme s ním dostali dobré výsledky. Zřejmě proto, že hráč pak vlastně upřednostňuje tahy, které mu propojí více souvislých oblastí.

```
.        .     % winrate [ .  / .   ] dt.rand  dt.sdc   dt.ste   dt.wpm_c xlogin00 xdufko02
dt.ste   45.69 % winrate [ 53 / 116 ] 52.1/48  46.2/52  45.7/116 43.3/60  43.1/72  45.7/116
dt.wpm_c 41.94 % winrate [ 52 / 124 ] 38.2/68  42.9/56  38.3/60  41.9/124 48.4/64  41.9/124
dt.sdc   27.59 % winrate [ 32 / 116 ] 30.0/60  27.6/116 26.9/52  19.6/56  32.8/64  27.6/116
xdufko02 18.00 % winrate [ 36 / 200 ] 21.6/116 19.8/116 11.2/116 18.5/124 18.8/128 18.0/200
dt.rand  13.79 % winrate [ 16 / 116 ] 13.8/116 13.3/60  14.6/48  10.3/68  17.9/56  13.8/116
xlogin00 8.59  % winrate [ 11 / 128 ] 10.7/56  7.8/64   8.3/72   7.8/64   8.6/128  8.6/128
```

Obrázek 2: Při prohledávání stavového prostoru jen do hloubky 1 už jsme překonali `dt.rand`.

Dále jsme pro ohodnocení stavu zkoušeli použít již existující funkci pro výpočet pravděpodobnosti udržení dané oblasti (`probability_of_holding_area()`), případně její kombinaci se skóre hráče, ale ani jeden přístup se neukázal jako dobrý.

Zvažovali jsme i další kritéria – např. celkový počet kostek hráče, ale .....

2.2 Problém v této fázi

Čas. S hloubkou prohledávání stavového prostoru větší než 1 jsme dostávali timeout.

3 ExpectiMiniMax – omezení 3 tahů

Jak algoritmus urychlit? Klíčová byla myšlenka, že hráč nemusí vždy chtít provést všechny možné tahy. Náš původní `ExpectiMiniMax` však možnost `EndTurnCommand()` neměl, dokud bylo co hrát, tak hrál.

Nově jsme tedy přidali omezení 3 tahů. Pokud už hráč udělal 3 tahy za sebou, ukončí svoje kolo. Tím se algoritmus zrychlil, bez timeoutu už jsme zvládli jít do hloubky 2 a výsledky se vylepšily.

Potvrdilo se, že někdy je výhodnější žádný tah nedělat. Tuto možnost jsme nezakomponovali přímo do prohledávání stavového prostoru, protože je dost ovlivněná náhodou. Museli bychom náhodně distribuovat kostky mezi pole hráče.

```
.         .     % winrate [ .  / .   ] dt.rand  dt.sdc   dt.ste   dt.wpm_c xlogin00 xdufko02
dt.ste    50.86 % winrate [ 59 / 116 ] 47.9/48  57.7/52  50.9/116 43.3/60  54.2/72  50.9/116
xdufko02  34.50 % winrate [ 69 / 200 ] 42.2/116 32.8/116 25.0/116 33.1/124 39.1/128 34.5/200
dt.wpm_c  28.23 % winrate [ 35 / 124 ] 29.4/68  35.7/56  21.7/60  28.2/124 26.6/64  28.2/124
dt.sdc    21.55 % winrate [ 25 / 116 ] 25.0/60  21.6/116 15.4/52  23.2/56  21.9/64  21.6/116
dt.rand   5.17  % winrate [ 6  / 116 ] 5.2/116  3.3/60   4.2/48   8.8/68   3.6/56   5.2/116
xlogin00  4.69  % winrate [ 6  / 128 ] 5.4/56   1.6/64   6.9/72   4.7/64   4.7/128  4.7/128
```

Obrázek 3: Výsledky se při hloubce prohledávání 2 zlepšily – algoritmus je výrazně lepší než varianta bez omezení tahů.

```
.         .     % winrate [ .  / .   ] dt.rand  dt.sdc   dt.ste   dt.wpm_c xlogin00 xdufko02
dt.wpm_c  38.71 % winrate [ 48 / 124 ] 38.2/68  32.1/56  43.3/60  38.7/124 40.6/64  38.7/124
dt.ste    37.07 % winrate [ 43 / 116 ] 39.6/48  30.8/52  37.1/116 35.0/60  41.7/72  37.1/116
xdufko02  31.00 % winrate [ 62 / 200 ] 36.2/116 32.8/116 25.9/116 26.6/124 33.6/128 31.0/200
dt.sdc    28.45 % winrate [ 33 / 116 ] 33.3/60  28.4/116 21.2/52  28.6/56  29.7/64  28.4/116
xlogin00  6.25  % winrate [ 8  / 128 ] 5.4/56   10.9/64  4.2/72   4.7/64   6.2/128  6.2/128
dt.rand   5.17  % winrate [ 6  / 116 ] 5.2/116  6.7/60   6.2/48   4.4/68   3.6/56   5.2/116
```

Obrázek 4: Při testu s hloubkou 1 jsou výsledky o něco horší, což jsme přesně čekali. Čím větší hloubka, tím přesněji předpovídáme ohodnocení stavu.

4 ExpectiMiniMax – urychlení výpočtu

Pokud chceme nasadit model na ohodnocování stavů, potřebujeme větší rychlost. Dalším krokem tedy bylo odstranění slabiny našeho `ExpectiMiniMax`, a to provádění deepcopy boardu při každém zanořování.

Při každém zanořování předpokládáme, že hráč `xdufko02` pole získal, tedy zkopírujeme board, provedeme daný krok (změníme vlastníka pole a přesuneme kostky). Pro tyto účely jsme vytvářeli hlubokou kopii, což však dlouho trvalo.

Novým přístupem bylo zavedení `do move` a `undo move`. Board nekopírujeme, máme jeden, při zanořování uděláme krok (změníme vlastníka pole a přesuneme kostky) a při vynořování ho zase vrátíme zpět (změníme vlastníka pole na původního a vrátíme kostky do původní pozice). Pseudokód je k nahlédnutí níže:

```
def expectiMiniMax(source, target, attack_prob, depth, board):
    if depth == 0:
        return get_score() * attack_prob
    do_move()
    nodes = possible_attacks()
    if len(nodes) == 0:
        score = get_score() * attack_prob
        undo_move()
        return score
    attack_probs = []
    nodes_scores = []
    for n in nodes:
        attack_probs.append(attack_success_probability())
        nodes_scores.append(expectiMiniMax(source, target,
        attack_probs[i], depth-1, board))
    undo_move()
    return max(nodes_scores) * attack_prob
```

Algoritmus se o dost urychlil, mohli jsme bez problémů vyzkoušet zanoření do hloubky 2 i 3. Větší zanoření ovšem nezlepšilo výsledky a do větší hloubky než 3 se nemá smysl zanořovat, protože jsme si nastavili omezení 3 tahů. Jakmile hráč udělá tři tahy za sebou, pak vrací `EndTurnCommand()`.

```
.         .     % winrate [ .   / .   ] dt.rand  dt.sdc   dt.ste   dt.wpm_c xlogin00 xdufko02
dt.ste    50.00 % winrate [ 58  / 116 ] 45.8/48  57.7/52  50.0/116 43.3/60  52.8/72  50.0/116
xdufko02  35.50 % winrate [ 71  / 200 ] 43.1/116 33.6/116 25.9/116 33.9/124 40.6/128 35.5/200
dt.wpm_c  28.23 % winrate [ 35  / 124 ] 29.4/68  35.7/56  21.7/60  28.2/124 26.6/64  28.2/124
dt.sdc    20.69 % winrate [ 24  / 116 ] 25.0/60  20.7/116 15.4/52  21.4/56  20.3/64  20.7/116
dt.rand   5.17  % winrate [ 6   / 116 ] 5.2/116  3.3/60   4.2/48   8.8/68   3.6/56   5.2/116
xlogin00  4.69  % winrate [ 6   / 128 ] 5.4/56   1.6/64   6.9/72   4.7/64   4.7/128  4.7/128
```

Obrázek 5: Výsledky při hloubce prohledávání 2 zlepšily zůstaly stejné, tj. kroky `do_move()` a `undo_move()` jsme implementovali správně.

```
.         .     % winrate [ .   / .   ] dt.rand  dt.sdc   dt.ste   dt.wpm_c xlogin00 xdufko02
dt.ste    47.41 % winrate [ 55  / 116 ] 54.2/48  38.5/52  47.4/116 50.0/60  47.2/72  47.4/116
xdufko02  33.50 % winrate [ 67  / 200 ] 38.8/116 35.3/116 26.7/116 28.2/124 38.3/128 33.5/200
dt.wpm_c  30.65 % winrate [ 38  / 124 ] 33.8/68  32.1/56  23.3/60  30.6/124 32.8/64  30.6/124
dt.sdc    24.14 % winrate [ 28  / 116 ] 28.3/60  24.1/116 21.2/52  25.0/56  21.9/64  24.1/116
xlogin00  6.25  % winrate [ 8   / 128 ] 1.8/56   9.4/64   5.6/72   7.8/64   6.2/128  6.2/128
dt.rand   3.45  % winrate [ 4   / 116 ] 3.4/116  5.0/60   2.1/48   2.9/68   3.6/56   3.4/116
```

Obrázek 6: Díky vyšší rychlosti jsme mohli vyzkoušet i hloubku 3. Lepší výsledky jsme ale nedostali.

5 Využitá literatura a zdroje

[1] Carol A. Luckhardt and Keki B. Irani. "An Algorithmic Solution of N-Person Games". In: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*. AAAI'86. Philadelphia, Pennsylvania: AAAI Press, 1986, pp. 158–162.