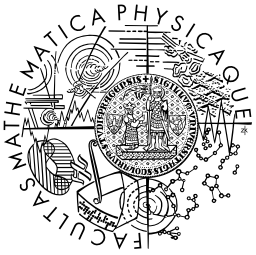


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Viktor Vašátko

Umělá inteligence pro hru Risk

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce:	Mgr. Martin Pilát, Ph.D.
Studijní program:	Informatika
Studijní obor:	Programování a Softwarové Systémy

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Rád bych poděkoval svému vedoucímu Mgr. Martinu Pilátovi, Ph.D. za odborné vedení práce, cenné rady, připomínky a ochotu projevenou při konzultaci. Dále bych poděkoval své rodině za podporu a trpělivost při mém studiu. Mé díky patří i všem dalším učitelům za získané vědomosti v průběhu studia.

Název práce: Umělá inteligence pro hru Risk

Autor: Viktor Vašátko

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D.,
Katedra teoretické informatiky a matematické logiky

Abstrakt: Cílem této práce je implementace hry Risk a průzkum, jak do hry vyvinout umělou inteligenci. Implementace hry a experimenty s umělou inteligencí jsou napsány v programovacím jazyce C#. V průzkumu vývoje umělé inteligence jsou otestovány dva přístupy. Jeden přístup využívá algoritmus Monte Carlo tree search (MCTS) s dvěma různými heuristikami a druhý se snaží řešit problém za pomocí neuronových sítí. Neuronové sítě mají dvojí využití. Jedno využití je pro samotnou umělou inteligenci a druhé v jedné z heuristik. Provedeným výzkumem jsme zjistili, že umělá inteligence využívající MCTS dokáže hrát nejlépe, ale nezvládá velké mapy. Oproti tomu umělá inteligence s neuronovými sítěmi dokáže hrát na libovolné mapě, a proto by mohla být potencionálně dalším cílem výzkumu. Výsledkem práce je základ pro další možný vývoj umělé inteligence.

Klíčová slova: deskové hry, umělá inteligence, Risk

Title: Artificial Intelligence for the Risk Game

Author: Viktor Vašátko

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D.,
Department of Theoretical Computer Science and Mathematical Logic

Abstract: The aim of this work is the implementation of the game Risk and exploration how to develop artificial intelligence into game. Game implementation and artificial intelligence experiments are written in C# programming language. Two approaches are tested in the development of artificial intelligence. One approach uses the Monte Carlo tree search algorithm (MCTS) with two different heuristics and the other tries to solve the problem with neural networks. Neural networks have two uses. One use is for the artificial intelligence itself and the second use is in one of the heuristics. Through the research, we found that the best player is the artificial intelligence using MCTS, but on big maps it has problem. On the other hand, the artificial intelligence with neural network can play on any map and therefore could potentially be another aim of research. The result of the work is the basis for further development of artificial intelligence.

Keywords: board games, artificial intelligence, Risk

Obsah

Úvod

Risk[1] je strategická desková hra, kde se hráči snaží ovládnout celý svět. Pomocí svých armád dobývají nepřátelská území, za která pak mohou získat novou armádu nebo risk karty. Pokud hráč získá určitou kombinaci karet, může ji vyměnit za další armádu a posílit své hranice. Risk je populární hra a má mnoho podob, ať už vzhledových - ztvárnění velkých herních nebo filmových titulů, nebo klasické dobývání celého světa. Hra má také různá pravidla.

Ve hře je velmi důležitá strategie a plánování. Je to časově náročná hra, a proto novým hráčům se Risk může zdát být náročný a zdlouhavý. Oproti jiným deskovým hrám, kdy každý hráč ve svém kole provede jeden tah, v Risku se kolo hráče skládá ze tří fází a hráč provede sérii tahů. Každá fáze ovlivňuje následující, takže hráč musí dopředu promyslet, jaký provede manévr.

Z hlediska umělé inteligence je hra ještě zajímavější. Kromě velkého významu dobré strategie a plánování má další důležité faktory, které činí určitým typům inteligencí velký problém. Jedním z faktorů je obrovské množství tahů, které za své kolo může hráč provést, takže prohledávání stavového prostoru bez silné heuristiky nelze. Risk má jeden z největších větvících faktorů. Dalším problémem je náhodnost. Souboje armád se rozhodují pomocí hodu kostek. Útočník může házet jednou až třemi kostkami - dle velikosti útoku a obránce, resp. jednou až dvěma kostkami - dle velikosti obrany. Díky větvícímu faktoru a náhodnosti je velmi obtížné vyvinout adekvátní umělou inteligenci pro tuto hru.

Cílem práce je nejprve naimplementovat hru Risk, která bude sloužit k experimentům s umělou inteligencí. Hra bude naimplementována v programovacím jazyce C#. Hru si může zahrát i uživatel. Bude mít na výběr ze singleplayeru a multiplayeru. V singleplayeru bude moci uživatel vyzvat jednoho z agentů a v multiplayeru si bude moci zahrát proti ostatním hráčům. Po implementaci hry je hlavním cílem prozkoumat, jak do hry vyvinout umělou inteligenci, a následně otestovat jednotlivé možnosti vývoje.

Kapitola 1 se zaměřuje na popis hry. Stručně vymezuje pravidla hry včetně různých modifikací. Podstatná část kapitoly je pak věnována herní strategii - obsazení kontinentů, rozestavění armád, útoku a obraně. Kapitola 2 je zaměřena na analýzu pravděpodobnosti útoku a obrany, analýzu kontinentů a stanovení složitosti stavového prostoru a větvícího faktoru. Podrobnou analýzu těchto faktorů posléze využijeme k vývoji umělé inteligence, aby dosahovala lepších výsledků. Kapitola 3 je věnována rozhraní hry, které je nutné pro snadnou a efektivní implementaci agentů. Budeme se zabývat komunikací mezi hráčem a hrou, její optimalizací a nakonec si popíšeme potřebné interfacy. V kapitole 4 prozkoumáme základní teorii k vytvoření různých druhů agentů a poté se pustíme do jejich implementace viz. kapitola 5. Celou naší práci uzavřeme kapitolou 6, kde otestujeme a porovnáme vytvořené agenty.

1. Hra Risk

1.1 Úvod

Risk[1] je desková strategická hra pro 3 – 6 hráčů, jejichž úkolem je v základní podobě hry ovládnout celou mapu a porazit ostatní hráče. Hráči postupně obsazují nová území, brání již získaná území, přemisťují své jednotky, dobývají celé kontinenty a snaží se o nadvládu nad celým světem. Existuje i modifikace hry pro dva hráče, případně původní verze hry, která dosažení výhry podmiňuje splněním určité mise.

Hra byla vytvořena francouzským režisérem, básníkem a spisovatelem Albertem Lamorissem[2]. Hru vymyslel na rodinné dovolené v Holandsku a nazval ji La Conquête du Monde (Dobytí světa). Následně v roce 1957 byla publikována francouzskou herní společností Miro. Do Ameriky se dostala přes Parker Brothers, kteří v 50. letech 20. století navázali vztah se společností Miro. V roce 1959 je publikována v Americe pod dnešním názvem Risk, ale její pravidla se v některých aspektech liší oproti původním. V Americe si užívali delší verzi hry, kdežto v Evropě hráli s tajnými misemi, kdy k vítězství stačilo splnit misi typu - dobuď Severní a Jižní Ameriku. Evropská verze se nakonec roku 1993 dostala i do Ameriky. Ne každá podoba Risku se těšila úspěchu. V roce 1986 vznikl Castle Risk. Hrálo se pouze na mapě Evropy a obsahoval skryté armády, karty s veliteli a špehy, hrady a námořnictvo. Tato verze hry se nestala příliš populární. V průběhu dalších let vznikaly různé podoby Risku zejména podle filmů. V roce 2001 vznikla Sci-fi podoba Risku, kdy hráči mohli dobývat měsíc a podmořský svět. Byli přidáni velitelé a hra se omezila na pět kol. Následovaly edice Risk: Star Wars nebo také Risk: Lord of Rings. Nakonec, v roce 2008 byla společností Hasbro vydána nová základní edice z roku 1959, která se v drobných obměnách prodává dodnes.

1.2 Součásti hry

Hra se skládá z:

- hrací desky
- 5 kostek – 2 bílé a 3 červené
- balíček 44 karet – v rozšířené verzi ještě dalších 14 karet s tajnými misemi
- 6 armád, každá s odlišnou barvou

Hrací deska je mapa obsahující 6 kontinentů rozdělených do 42 území. Každý kontinent má odlišnou barvu a skládá se ze 4 – 12 území. Na hrací desce jsou dále uvedeny počty armád, které hráč obdrží za výměnu kombinace karet a za ovládnutí kontinentu.

Každá armáda se skládá ze 3 typů jednotek – pěchota, kavalérie, dělostřelectvo. Jednotlivé typy jednotek představují určitou velikost armády. Pěchota odpovídá velikosti 1 armády, kavalérie velikosti 5 armád a dělostřelectvo velikosti

10 armád. Hráči v průběhu hry umísťují jednotky na území a v případě potřeby mohou vyměnit svou jednotku za jinou, ale musí zachovat velikost armády na území. Hráč může například vyměnit 5 jednotek pěchoty za 1 kavalérii atd.

Balíček karet obsahuje 42 normálních karet, kdy každá karta je označena územím a typem jednotky, další dva žolíky, kteří jsou označeni všemi třemi typy jednotek.

1.3 Pravidla hry

1.3.1 Světová nadvláda

Cílem hry je získat nadvládu nad celým světem.

Začátek hry

Každý hráč si vybere barvu svých jednotek a podle počtu hráčů pak každý hráč obdrží určitý počet armád pro začátek hry.

- 3 hráči – každý obdrží 35 armád
- 4 hráči – každý obdrží 30 armád
- 5 hráčů – každý obdrží 25 armád
- 6 hráčů – každý obdrží 20 armád

Po získání pěchot všichni hráči hodí jednou kostkou a ten, kdo hodí nejvyšší číslo, začíná hrát. Umístí jednu armádu na neobsazené území a poté pokračuje hráč nalevo. Hráči pokračují tak dlouho, dokud neobsadí všech 42 území. Po obsazení všech území hráči pokračují v rozmísťování, kdy mohou přidat pěchotu na libovolné jimi obsazené území.

Jakmile je všech 42 území obsazeno armádami jednotlivých hráčů, zamíchá se balíček karet a pak hráč, který začínal s rozmísťováním jednotek, začíná hrát.

Průběh hry

Tah hráče obsahuje 3 fáze, kterými prochází přesně v tomto pořadí – rozmístění nové armády, útok a přemístění armády. Hráč se snaží obsadit území protihráčů poražením jeho armády, ale zároveň musí přemýšlet dopředu a myslet na obranu a posílení hranic.

Na začátku první fáze si hráč spočítá, kolik obdrží nových armád na základě:

- obsazených území
- ovládnutých kontinentů
- směněných kombinací karet
- označených území na směněných kartách

Hráč si spočítá všechna svá obsazená území a výsledek celočíselně vydělí třemi (ignoruje zlomky). Výsledek určí, kolik dostane armád. Platí, že vždy obdrží minimálně 3 armády.

Při počítání obsazených území se hráč podívá, zda neovládá jeden z kontinentů. Pokud ovládá jeden nebo více kontinentů, získává armádu navíc, jejíž velikost se odvíjí od kontinentu. Armáda získaná za ovládnutí kontinentů je následující:

- Severní Amerika – 5 armád
- Jižní Amerika – 2 armády
- Afrika – 3 armády
- Evropa – 5 armád
- Asie – 7 armád
- Austrálie – 2 armády

V průběhu první fáze může hráč získat ještě další armády vyměněním kombinace karet. Každá kombinace se skládá ze 3 karet. Ve hře lze rozměnit 3 typy kombinací karet. První kombinace se skládá ze stejného typu jednotek, takže hráč může rozměnit např. 3 karty s označením pěchota. Druhou kombinaci karet tvoří 3 karty, přičemž každá z nich musí být označena jiným typem jednotky - pěchota, kavalérie, dělostřelectvo. Poslední kombinaci tvoří libovolné dvě karty doplněné žolíkem.

Při výměně karet obdrží hráč armády podle toho, kolikátá kombinace karet byla rozměněna:

- první kombinace karet – 4 armády
- druhá kombinace karet – 6 armád
- třetí kombinace karet – 8 armád
- čtvrtá kombinace karet – 10 armád
- pátá kombinace karet – 12 armád
- šestá kombinace karet – 15 armád
- každá další kombinace karet – vždy se přičte dalších 5 armád navíc

Pokud na jedné ze směňovaných karet bylo označené území, které hráč vlastní, obdrží další dvě armády, Tyto armády přitom musí být umístěny na území označené na kartě. Hráč nesmí nikdy mít v ruce více než 5 karet. Pokud hráč má 5 nebo 6 karet, musí je vyměnit za armády. Po výměně karet a rozmístění armád první fáze končí a přichází útok.

Fáze útoku není povinná. Pokud hráč v tento moment nechce útočit, nemusí a může rovnou přejít k fázi přesunu jednotek, případně rovnou předat tah dalšímu hráči. Na útok by měl hráč myslet již v první fázi tahu, kdy rozmísťuje jednotky. Cílem útoku je obsadit nepřátelská území. Toho je docíleno poražením všech nepřátelských jednotek vyskytujících se na daném území. Souboj probíhá pomocí

házení kostek. Při útoku si nejprve hráč vybere území, ze kterého bude útočit. Toto území musí sousedit s nepřátelským územím, na které chce hráč zaútočit, a musí mít alespoň 2 armády. Hráč může útočit tak dlouho, dokud splňuje výše uvedenou podmínku a může útočit i na jiná území. Nemusí útočit jen na jedno. Celý útok na území je řízen hodem kostkami. Obránce si vezme bílé kostky. Pokud má alespoň 2 armády na svém území, vezme si dvě bílé kostky, jinak si vezme jednu bílou kostku. Útočník si může vybrat s jak velkým útokem vyrazí. S jednou armádou bude házet jednou kostkou, s dvěma armádami bude házet dvěma kostkami a s třemi armádami bude házet třemi kostkami. Avšak vždy musí zůstat alespoň jedna armáda na jeho území. Například útočník má na území 3 armády, pak jeho největší útok může být s dvěma armádami, protože jedna musí zůstat na jeho území. Po provedeném hodu příslušným počtem kostek jsou hody porovnány. Vždy se porovnává nejvyšší hozené číslo s nejvyšším hozeným číslem soupeře. Pokud má útočník vyšší číslo, obránce ztratil armádu. Jinak, i když se čísla rovnají, ztratil armádu útočník. Za předpokladu, že obránce házel dvěma kostkami, pokračuje se dál a porovnávají se druhá nejvyšší čísla stejným způsobem. Čím větší armádou útočník útočí, tím větší má šanci na poražení obránce. Obránce nemůže porazit více armád, než kolika hází kostkami ani nemůže přijít o více armád.

Pokud útok proběhl úspěšně a útočník porazil všechny obráncovy armády, dochází k obsazení území, do kterého musí být ihned přesunuty armády. Hráč může přesunout armád, kolik chce, ale nejméně tolik s kolika armádami útočil. Mít většinu armády na hranicích s nepřítelem je spíše výhodou, protože armády za hranicemi při obraně nepomohou. Platí však, že vždy musí zůstat alespoň jedna armáda na každém obsazeném území. Pokud hráč obsadil alespoň jedno území, může si vzít z vrchu balíčku jednu kartu.

Po útoku přichází poslední fáze - přesun armád. Hráč na konci svého kola může přesunout libovolný počet armád z jednoho území do dalšího svého území, které je propojené skrze jím obsazenými územími. Tento přesun může provést právě jednou. Tím končí celé kolo hráče a hraje další hráč po levici.

Jestliže hráč během svého kola porazí jednoho z nepřátel, obdrží všechny jeho karty. Pokud tímto bude mít v ruce více než 6 karet, musí je vyměňovat tak dlouho, dokud nebude mít v ruce 4 a méně karet. Jakmile má 4, 3 karty, musí ihned přestat s vyměňováním. Pokud tímto bude mít 6 karet, musí počkat do dalšího kola. Po vyměnění karet, umístí získanou armádu a může pokračovat dál.

Hra končí v okamžiku, kdy jeden z hráčů ovládne celý svět.

1.3.2 Tajné mise

Jedná se o původní evropskou podobu hry, přičemž cílem hry je splnit tajnou misi, kterou hráč obdrží na začátku hry. V současnosti existuje 14 tajných misí, v předešlých verzích bylo misí 12. Mise jsou následující:

- Znič hráče s danou barvou (6 karet)
- Ovládni určitou kombinaci kontinentů
  - Ováldni Asii a Jižní Ameriku

  - Ovládni Asii a Afriku
  - Ovládni Severní Ameriku a Afriku
  - Ovládni Severní Ameriku a Austrálii
  - Ovládni Evropu, Jižní Ameriku a třetí kontinent dle vlastní volby
  - Ovládni Evropu, Austrálii a třetí kontinent dle vlastní volby
- Obsaď 18 území alespoň 2 armádami
- Obsaď 24 území (bez určení počtu armád na územích)

Začátek hry

Opět jako v pravidlech Světové nadvlády, hráči na začátku obdrží určitý počet armád podle počtu hrajících hráčů.

- 3 hráči – každý obdrží 35 armád
- 4 hráči – každý obdrží 30 armád
- 5 hráčů – každý obdrží 25 armád
- 6 hráčů – každý obdrží 20 armád

Než se přejde k další části, hráči si zvolí jednoho hráče, který se na chvíli stane generálem. Pokud hraje méně než 6 hráčů, generál odstraní neplatné tajné mise z balíčku. Zamíchá tajné mise a rozdá každému hráči jednu misi, začne hráčem po levé ruce. Nikdo, včetně generála, se nesmí podívat na svou tajnou misi. Dále generál vezme balíček risk karet, odstraní dva žolíky a rozdá všechny karty, opět přitom začíná po své levici. Tyto rozdané karty určí počáteční území hráčů a každý z hráčů na svá území položí právě jednu armádu. V případě, že hraje 4 nebo 5 hráčů, 2 hráči obdrží kartu navíc oproti ostatním. Poté hráči pokračují už libovolně v umísťování armád na svá území. Nakonec generál vezme všechny risk karty, přidá žolíky, zamíchá je, každý se podívá na svou tajnou misi a začíná se hrát.

Průběh hry

Průběh hry je stejný jako ve Světové nadvládě. Hraje se tak dlouho, dokud jeden z hráčů nedokončí misi a neukáže svou tajnou misi. V případě, že někdo má za úkol v rámci mise zničit určitého hráče, může se stát, že jiný hráč porazí poslední armádu daného hráče a tím pomůže hráči s touto misí k vítězství.

1.3.3 Modifikace pro 2 hráče

Cílem hry je porazit protihráče. Oproti původním verzím pravidel, jsou zde i neutrální území. Ta však nemohou dobývat ostatní území, pouze se mohou bránit. Je to taková náhrada třetího hráče.

Začátek hry

Oba hráči si vyberou barvu jednotek, pak zvolí třetí barvu, kterou prohlásí za neutrální. Každý si vezme 40 armád, neutrálnímu hráči dají rovněž 40 armád. Z balíčku risk karet se odstraní žolíci, zbylé karty se zamíchají a vytvoří se náhodně tři stejně velké hromádky karet. Každý hráč si vybere hromádku, třetí zbude neutrálnímu hráči. Podle vyznačených území na svých kartách každý hráč umístí právě jednu armádu na tato území a pak se umístí i armády neutrálního hráče. Zbylou armádu hráči umísťují libovolně, vždy každý umístí 2 armády libovolně a jednu armádu od neutrálního hráče tak, aby omezil protihráče. Po rozmístění armád se vrátí žolíci do balíčku karet, karty se zamíchají a hra začíná.

Průběh hry

Průběh hry je stejný jak ve Světové nadvládě s tím rozdílem, že hráč může útočit i na neutrální území. Pokud se hráč rozhodne dobýt neutrální území, tak protihráč hází kostkami za neutrálního hráče. Neutrální hráč nemůže útočit, pouze se brání. Vyhrává ten, kdo zničí všechny protihráčovy armády.

1.3.4 Modifikace pravidel

Existuje několik dodatečných modifikací, které hru ztíží případně ji zkrátí. Jedna modifikace se týká výměny karet za armádu. Místo toho aby obdržené množství armád rostlo, zůstává konstantní. Za každý typ vyměněné kombinace hráč obdrží určitý počet armád, který se nemění v průběhu hry. Existuje také modifikace, která růst obdržených armád jen zpomaluje. První výměna kombinace karet je za 4 armády a následující jsou vždy o 1 větší oproti předchozí výměně. Kromě pravidel týkajících se karet, existuje také pravidlo upravující poslední fázi tahu hráče. Normálně mohou hráči přemístit armádu pouze z jednoho území do druhého, ale toto pravidlo povoluje přemístit armádu z jednoho nebo více území do jednoho nebo více území. Další možné pravidlo říká, že hráč může mít na jednom území maximálně 12 armád a pokud je na každém území 12 armád, nová armáda propadá.

1.4 Strategie

1.4.1 Základní strategie

Než zmíníme nějakou konkrétní strategii, měli bychom se podívat na základní strategii, která dopomůže k útoku a obraně. Hráč by se měl snažit soustředit svou armádu na hranicích. Armáda v centrálním území nijak nepomůže při obraně ani při obsazování dalších území. Dále by měl hráč sledovat okolní dění. Pokud nepřítel někde centralizuje armádu, měl by se připravit na útok. Naopak, pokud nepřítel obsazuje nějaký kontinent a je to v hráčově moci, měl by mu obsazení kontinentu překazit, aby nepřítel nezískal bonusy. Nakonec by se hráč měl snažit obsadit nějaký kontinent, aby získal bonusy. Obsazení kontinentu by však měl brát s rozvahou. Pokud není schopen kontinent udržet po dobu alespoň jednoho kola, nemá takové obsazení význam a spíše se stane snadným terčem a velmi se oslabí.

1.4.2 Obsazení kontinentů

Důležitou otázkou je, jaký kontinent bychom měli obsadit, abychom měli co největší výhodu a silný základ. Někteří hráči tvrdí, že nejlepší je obsadit Austrálii, protože má na hranici pouze jedno území a dobře se brání. Další říkají, že je dobrá Asie, protože má největší bonus, ale každá z možností má určitou nevýhodu. Austrálie má nevýhodu, že je izolovaná a hráč těžko rozšiřuje svá území. Austrálie má dobrou pozici z obranného hlediska, ale nakonec se hráč nemá šanci ubránit proti jinému, který obsadil půlku světa. Oproti tomu Asie není izolovaná, ale většina území je na hranicích a těžko se brání. Zkusme se podívat na kontinent Severní Amerika[3].

Severní Amerika má největší bonus v poměru počtu území a velikosti hranic. Další výhodou je snadná expanze. Ze Severní Ameriky se lze dostat do Jižní Ameriky, Evropy a Asie. Tímto způsobem hráč lehce překazí ovládnutí těchto kontinentů a ostatní připraví o bonusy. Až si hráč nastřádá trošku více armády, může hned převzít Jižní Ameriku a stále bude mít jen 3 území na hranici. Takhle bude mít jeden z největších bonusů. Nakonec nejprve obsadí Afriku spolu s Evropou a v tuto chvíli už je ve výherní pozici. Asie se těžko brání a zbývající Austrálie, i když má dobrou obranu, nemá šanci proti hráči s tak velkými bonusy.

V některých případech se nehodí soustředit na kontinenty. V případě, kdy protihráči mají dobře bráněné hranice, kontinent je moc velký nebo by to moc oslabilo hráče, se hodí soustředit na lehké obsazení více území, než zvolit útok s velkým odporem a obsazení pár území z kontinentu. Pokud hráč obsadí celkově 6 území místo toho, aby se snažil dobýt Jižní Ameriku, kde je velký odpor, bude na tom lépe. Potom až obsadí větší množství území, bude získávat více armády a může zaútočit na dobře bráněné kontinenty.

1.4.3 Rozestavění armády

Na začátku hry, kdy hráči obsazují svou armádou neobsazená území, je dobré se snažit obsadit nějaký kontinent. To se pravděpodobně nepovede, protože protihráči se budou snažit tento záměr překazit. Platí však, že čím více obsazených území bude mít hráč na jednom kontinentu, tím lépe se mu dobude celý kontinent. Navíc je dobré udržovat území blízko sebe. To ale neznamená, že další skupinku území nemůžeme mít někde dál. V některých případech to může být výhodou a také by hráč neměl zapomenout na blokování obsazení kontinentu protihráčem.

1.4.4 Útok a obrana

I když se útok a obrana řídí hody kostkami, neznamená to, že hráč nemůže nijak ovlivnit úspěšnost svého postupu. Obránce může házet vždy buď jednou nebo dvěma kostkami. Pokud hází jednou a hráč jde s velkým útokem, obránce je ve velké nevýhodě. Pravděpodobnost, že útočník hodí vyšší číslo, je větší než pravděpodobnost, že ho hodí obránce. V případě, že obránce bude házet dvěma kostkami a útočník bude házet třemi, obránce je opět v nevýhodě. Z toho plyne, že útočit je daleko výhodnější než bránit, pokud útočník jde s dostatečně velkou armádou. Hráč by neměl být moc defenzivní, ale spíše agresivní. I v této hře platí, že nejlepší obrana je útok. Když se hráč rozhodne útočit, měl by útočit s cílem převzít území a bojovat až do doby, než území převezme, nebo se příliš oslabí,

nebo oslabit přicházející útok. Jinak útok nemá smysl a spíše se hráč zbytečně oslabí. Při útoku musí hráč vzít v úvahu velikost své armády a armády obránce. Čím více armády útočník má, tím pravděpodobnější je dobytí obráncova území.

2. Analýza

2.1 Pravděpodobnosti útoku a obrany

I když hody kostkou jsou náhodné, lze spočítat, že v některých případech je pravděpodobnější vítězství útočníka a někdy obránce. Tyto poznatky se nám budou hodit při vyvíjení lepší umělé inteligence. Nejprve se pojďme podívat na pravděpodobnosti všech možných hodů kostkami. Jediné možné hody útočník-obránce jsou 1-1, 2-1, 3-1, 1-2, 2-2, 3-2. Nadefinujeme dva jevy (viz Tabulka 2.1).

O_x	Obránce ztratil x armád
U_x	Útočník ztratil x armád

Tabulka 2.1: Definice jevů

Obránce má 1 kostku			
Počet útočníkových kostek	1	2	3
$\mathsf{P}(O_1)$	41,67%	57,87%	65,97%
$\mathsf{P}(U_1)$	58,33%	42,13%	34,03%

Tabulka 2.2: Hody kostkou 1

Obránce má 2 kostky			
Počet útočníkových kostek	1	2	3
$\mathsf{P}(O_1)$	25,46%	N/A	N/A
$\mathsf{P}(U_1)$	74,54%	N/A	N/A
$\mathsf{P}(O_2)$	N/A	22,76%	37,17%
$\mathsf{P}(U_2)$	N/A	44,83%	29,26%
$\mathsf{P}(O_1 \wedge U_1)$	N/A	32,41%	33,58%

Tabulka 2.3: Hody kostkou 2

Z pravděpodobností[4] (viz Tabulka 2.2 a 2.3) můžeme vidět, že ať už obránce hází jednou nebo dvěma kostkami, pokud útočník hází třemi kostkami, má více než 60% šanci, že zničí útočníkovi alespoň jednu armádu. Útočník, který má na území alespoň 4 armády, má výhodu oproti obránci. Tyto poznatky nám napovídají, že expanzivní hraní je výhodnější než defenzivní, takže by agent měl hrát agresivněji, aby zvítězil. Pojďme se podívat, jak dopadnou souboje mezi více armádami a ne jen při jednom hodu kostkami, abychom věděli, kdy se vyplatí útočit a kdy raději ne (viz Tabulka 2.4 a 2.5). Počty armád útočníka znamenají všechny armády, které mohou útočit. Na území útočníka musí být o jednu armádu navíc, protože jedna armáda musí zůstat na území.

Armáda obránce \ Armáda útočníka	1	2	3	4	5
1	41.67%	75.42%	91.64%	97.15%	99.03%
2	10.61%	36.27%	65.60%	78.55%	88.98%
3	2.70%	20.61%	47.03%	64.16%	76.94%
4	0.69%	9.13%	31.50%	47.65%	63.83%
5	0.18%	4.91%	20.59%	35.86%	50.62%
6	0.05%	2.14%	13.37%	25.25%	39.68%
7	0.01%	1.13%	8.37%	18.15%	29.74%
8	0,00%	0.49%	5.35%	12.34%	22.41%
9	0.00%	0.26%	3.28%	8.62%	16.16%
10	0.00%	0.11%	2.08%	5.72%	11.83%

Tabulka 2.4: Pravděpodobnosti úspěchu útoku 1

Armáda obránce \ Armáda útočníka	6	7	8	9	10
1	99,67%	99,88%	99,96%	99,99%	100,00%
2	93,40%	96,67%	98,03%	99,01%	99,42%
3	85,69%	90,99%	94,68%	96,70%	98,11%
4	74,49%	83,37%	88,78%	92,98%	95,39%
5	63,77%	73,64%	81,84%	87,29%	91,63%
6	52,07%	64,01%	72,96%	80,76%	86,11%
7	42,33%	53,55%	64,29%	72,61%	79,98%
8	32,95%	44,56%	54,73%	64,64%	72,40%
9	25,78%	35,69%	46,40%	55,81%	65,01%
10	19,34%	28,68%	37,99%	47,99%	56,76%

Tabulka 2.5: Pravděpodobnosti úspěchu útoku 2

Na základě těchto údajů se dozvídáme, že pokud má obránce větší nebo stejný počet armád na daném území, je menší pravděpodobnost, že ho útočník dobude a nemá cenu útočit. Zajímavé je, že s rostoucím počtem útočníkovy armády roste i pravděpodobnost dobytí území obránce se stejným počtem armád jako útočník útočí. Už od pěti armád získává výhodu útočník. Opět se potvrdilo, že defenzivní hraní není moc vhodné, protože pravděpodobnost úspěchu útoku roste velice rychle s rostoucím počtem armád. Z tohoto pozorování lze vyvodit, že výsledky soubojů nejsou tak náhodné, jak se na začátku jevilo.

2.2 Analýza kontinentů

Víme-li, jaká je pravděpodobnost výsledku souboje a jsme schopni určit, jestli souboj vyhrajeme nebo ne, můžeme se podívat na bonusy kontinentů a jejich výhody a nevýhody. Obsazení dobrého kontinentu nám může poskytnout dobrou základnu a zvýšit pravděpodobnost vítězství. Podívejme se na základní údaje o kontinentech (viz Tabulka 2.6).

	Evropa	Asie	Austrálie	Afrika	J. Amerika	S. Amerika
Území	7	12	4	6	4	9
Hranice	4	5	1	3	2	3
Bonus	5	7	2	3	2	5

Tabulka 2.6: Základní informace o kontinentech

Na první pohled to vypadá, že Asie je nejlepší kontinent, protože má největší bonus. Pojďme se podívat blíže na vlastnosti jednotlivých kontinentů. Kromě bonusu za kontinent, je důležitým faktorem, kolik území je potřeba za jednu armádu, jak velké má hranice, kolik armády padne na jedno území na hranici a jaký je obranný koeficient (viz Tabulka 2.7).

Definice 1. *Nechť x_1 je počet území na kontinentu, x_2 je počet území na hranici, x_3 počet území, které mohou útočit na kontinent, b bonus za kontinent a y množství armády za jedno území. Pak definujeme následující veličiny:*

Počet armád za ovládnutí kontinentu.

$$z_0 = yx_1 + b$$

Počet území za jednu armádu.

$$z_1 = \frac{x_1}{z_0}$$

Počet armád na jedno území na hranici.

$$z_2 = \frac{z_0}{x_2}$$

Obranný koeficient nebo-li poměr počtu území, které mohou útočit na kontinent a území na hranici kontinentu.

$$z_3 = \frac{x_3}{x_2}$$

Kontinent	z_0	z_1	z_2	z_3
Evropa	7,33	0,95	1,83	1,5
Asie	11	1,09	2,2	1,2
Austrálie	3,33	1,2	3,33	1
Afrika	5	1,2	1,67	1,33
J. Amerika	3,33	1,2	1,67	1
S. Amerika	8	1,13	2,67	1

z_0... počet armád za kontinent
z_1... počet území za jednu armádu
z_2... počet armád na jedno území na hranici
z_3... obranný koeficient

Tabulka 2.7: Rozšířené informace o kontinentech

Po analýze zmíněných faktorů se Asie už nezdá být moc vhodným kontinentem na první obsazení. Má příliš velké hranice, těžce se brání a je potřeba obsadit mnoho území, aby hráč získal bonus. Evropa je na tom s obranou ještě hůře než

Asie, má nejtěžší obranu, ale není třeba obsadit tolik území, aby hráč získal bonus. Podobně je na tom Afrika. Nejlépe se dá ubránit Austrálie, ale moc nepomůže hráči s růstem armády. Navíc je Austrálie izolovaná od okolního světa, takže expanze na další území je velice náročná. Nejlepším kandidátem na první obsazení je nakonec Severní Amerika. Dobře se dá ubránit, bonus za kontinent je dostatečně velký a z tohoto kontinentu lze lehce expandovat do tří dalších kontinentů. Severní Amerika poskytuje nejvýhodnější startovací pozici a měla by být prvním cílem.

2.3 Složitost stavového prostoru

Definice 2. *Složitost stavového prostoru definujeme jako počet všech možných validních stavů hry.*

Hra Risk nemá žádná omezení. Hráči mohou mít na území libovolný počet armád, ale alespoň jednu musí mít na každém území. Na začátku hry hráči pouze zaplňují všechna území a pak doplňují zbylou armádou. V průběhu hry, každý hráč projde fází doplnění armády, útokem, který není povinný a přesunem armád, který také není povinný. To znamená, že hráči mohou klidně jen doplňovat armádu a nemusí útočit. Díky tomu dostáváme nekonečný stavový prostor. Nekonečný stavový prostor může být velkou komplikací pro některé umělé inteligence. Dokonce umělé inteligence nemusí dohrát hru v rozumném čase. Pokud budou příliš defenzivní může hra dojít do těžko dokončitelného stavu hry.

2.4 Větvící faktor

Definice 3. *Větvící faktor definujeme jako počet všech možných validních stavů, do kterých lze přejít ze současného stavu pomocí nějakého validního tahu.*

Problémem Risku je, že tahem se rozumí odehrání všech fází - od doplnění jednotek, přes útok až po přesun armády. Tah hráče pak tvoří série podtahů. Každý podtah nás dostane do určitého stavu, ze kterého opět můžeme provést další podtahy. Zkusme spočítat hrubý odhad všech možných kombinací tahů, které mohou následovat po určitém stavu hry, abychom měli představu o tom, jak náročné by bylo prohledávat všechny možné stavy.

Předpokládejme, že všichni tři hráči už rozestavili svoje armády a začínají bojovat. Máme klasickou mapu, která obsahuje 42 území, přičemž každý obsadí 14 území. Řekněme, že všichni hráči se snažili na každé území dát stejný počet armád, pak téměř na každém území jsou 3 armády. Hráči obsazovali území úplně náhodně, takže je mají rozmístěné po celé mapě. V průměru může každé území útočit na další tři okolní území. Hráč na začátku svého tahu obdrží 4 armády za obsazená území a může je doplnit kamkoliv na svá území. Má 14 území a 4 armády, všechny možné tahy nám udává kombinace s opakováním.

$$\binom{N+k-1}{k}$$

$$\binom{14+4-1}{4} = \binom{17}{4} = 2380$$

Po rozmístění armády může hráč útočit nebo přejít k další fázi. Řekli jsme, že může útočit se všemi 14 územími a každé může útočit na 3 území vedle sebe. Při útoku na jedno území je 22 různých kombinací podtahů. Spočítejme, že s daným územím si vybere jestli chce útočit a pokud ano, tak si vybere jedno ze tří nepřátelských území. To nám dává 66 možných tahů pro jedno území, ale toto může udělat u každého území a těch je 14. Pak dostáváme následující výsledek všech možných tahů po dvou fázích.

$$\binom{17}{4} 66^{14} \approx 10^{29}$$

Větvící faktor je obrovský na to, abychom mohli prohledávat všechny stavy a pamatovat si je. Agenty prohledávající stavový prostor nebudeme moci použít bez velmi silné heuristiky. Silná heuristika pak ale může odstranit tahy, které by mohly být pro určitý stav hry dobré. Navíc kvalitní heuristika může být náročná k vyvinutí, aby agent hrál opravdu dobře.

Již Michael Wolf ve své diplomové práci „An Intelligent Artificial Player for the Game of Risk"[5] zmínil porovnání složitostí her. Na tomto porovnání jde krásně vidět, proč je hra Risk tak náročná oproti jiným (viz Tabulka 2.8). Místo větvícího faktoru využívá složitost herního stromu.

Definice 4. *Nechť VF je větvící faktor a PPK je průměrný počet kol, pak složitost herního stromu definujeme jako VF^{PPK}.*

Hra	Složitost stavového prostoru	Složitost herního stromu
Mlýn	10^{10}	10^{50}
Dáma	10^{18}	10^{31}
Othello/Reversi	10^{28}	10^{58}
Šachy	10^{46}	10^{123}
Shogi	10^{71}	10^{226}
Go	10^{172}	10^{360}
Risk	∞	10^{5945}

Tabulka 2.8: Složitosti jednotlivých her

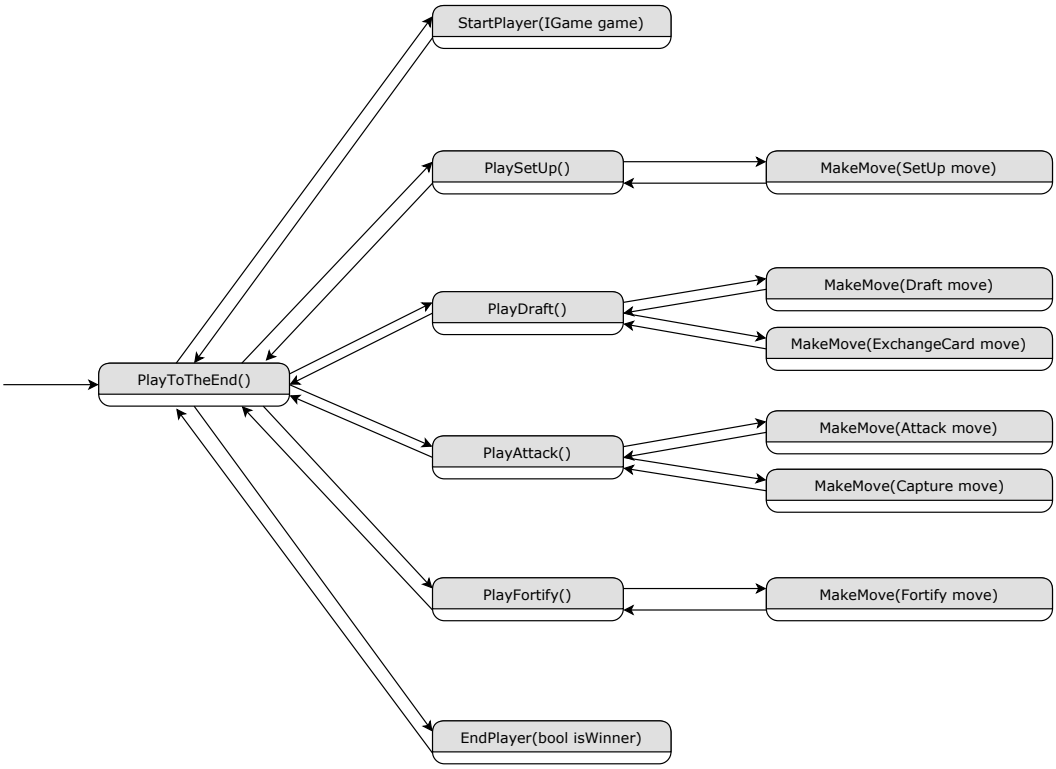
Uvedené herní složitosti stromu u ostatních her, kromě Risku, jsou maximální. U Risku je uvedená průměrná složitost herního stromu, protože maximální složitost herního stromu Risku je nekonečno. Dokonce i průměrná hodnota vysoce přesahuje hodnoty ostatních her.

3. Rozhraní hry Risk

V této kapitole se podíváme na návrh rozhraní hry Risk, který potřebujeme pro otestování a souboj agentů. Potřebujeme jednoduché a efektivní rozhraní, které nám umožní snadno připojit a naimplementovat agenta do hry. Komunikace mezi hrou a agentem musí být rychlá, aby nebrzdila celý průběh hry. Tento požadavek nám znemožňuje použití síťové komunikace, při které by mohlo docházet ke zpoždění. Další problém vyskytující se při síťové komunikaci je serializace. I když knihovny pro serializaci jsou dnes velice rychlé, bylo by to znatelné zpomalení. Pro lidské hráče toto zpomalení není znatelné, ale pokud budeme chtít odsimulovat hru agentů, docházelo by ke zpomalení simulace. Proto agent musí být co nejblíže hře, aby jejich komunikaci nic nebrzdilo.

Pro naši situaci se nejoptimálněji jeví komunikace dvou interfaců, která umožňuje implementaci lidského hráče pomocí síťové komunikace i agenta, který komunikuje přímo skrze volání metod. Pojďme se podívat na návrh interfacu hráče. Hráč potřebuje dostávat informace, jak vypadá herní plán, začátek hry, konec hry, výsledek, aktualizace herního plánu, kdy je hráč na tahu a jaké tahy má provést. Všechny informace lze předávat pomocí volání metod a jejich parametrů. Začátek hry, kdy hráč dostane informaci o tom, jak vypadá herní plán, budeme signalizovat metodou StartGame(IGame game). V parametru při zavolání metody hráč dostane objekt hry implementující interface IGame, který mu umožní si zjistit všechny potřebné informace o hře. Hráč si musí objekt uchovat, aby později mohl provádět tahy pomocí volání metod MakeMove(...). Až si každý hráč zjistí informace o hře, přejde se do úvodu hry, kdy si hráči rozestavují armády. Každý ve svém kole může vybrat právě jedno území, kde se umístí jedna armáda. Pokud nejsou všechna území obsazena, musí vybrat území neobsazené. Po zaplnění všech území doplňují hráči armády do jimi obsazených území. Hráči se předá kolo pomocí zavolání metody PlaySetUp(). Hráč v metodě vybere svůj tah a vytvoří jeho reprezentaci - objekt SetUp. Tah provede zavoláním metody MakeMove(SetUp move) na objektu hry. Po rozestavění armád přichází samotný souboj. Nyní se kolo hráče skládá ze tří fází: doplnění armády, útok, přemístění armády. Na základě toho se opět na hráči volají postupně metody: PlayDraft(), PlayAttack(), PlayFortify(). V metodě PlayDraft() musí hráč doplnit všechny volné armády. Pokud má hráč více než 5 karet, musí jich vyměnit tolik, aby snížil počet na méně než 5. Pro doplnění armády na určité území musí hráč vytvořit objekt Draft, který předá metodě MakeMove(Draft move), kterou zavolá na objektu hry. Úplně stejně vymění kombinaci karet. Hráč vybere kombinaci karet, vytvoří objekt ExchangeCard a zavolá metodu MakeMove(ExchangeCard move). Po ukončení první fáze hra zavolá na hráči metodu PlayAttack(). Hráč může, ale nemusí útočit. Útok reprezentuje objekt Attack, který jako parametr bere metoda MakeMove(Attack move). Tady musí hráč kontrolovat, zda dobyl území, nebo dokonce nevyhrál. Po dobytí území musí ihned provést tah obsazení vytvořením objektu Capture a zavoláním MakeMove(Capture move). Pokud hráč vyhrál, stačí když ukončí fázi útoku. V poslední fázi se zavolá metoda PlayFortify(), kde hráč může, ale nemusí, provést jedno přemístění. K tomuto tahu slouží objekt Fortify a metoda MakeMove(Fortify move). Až jeden z hráčů vyhraje, na všech hráčích se

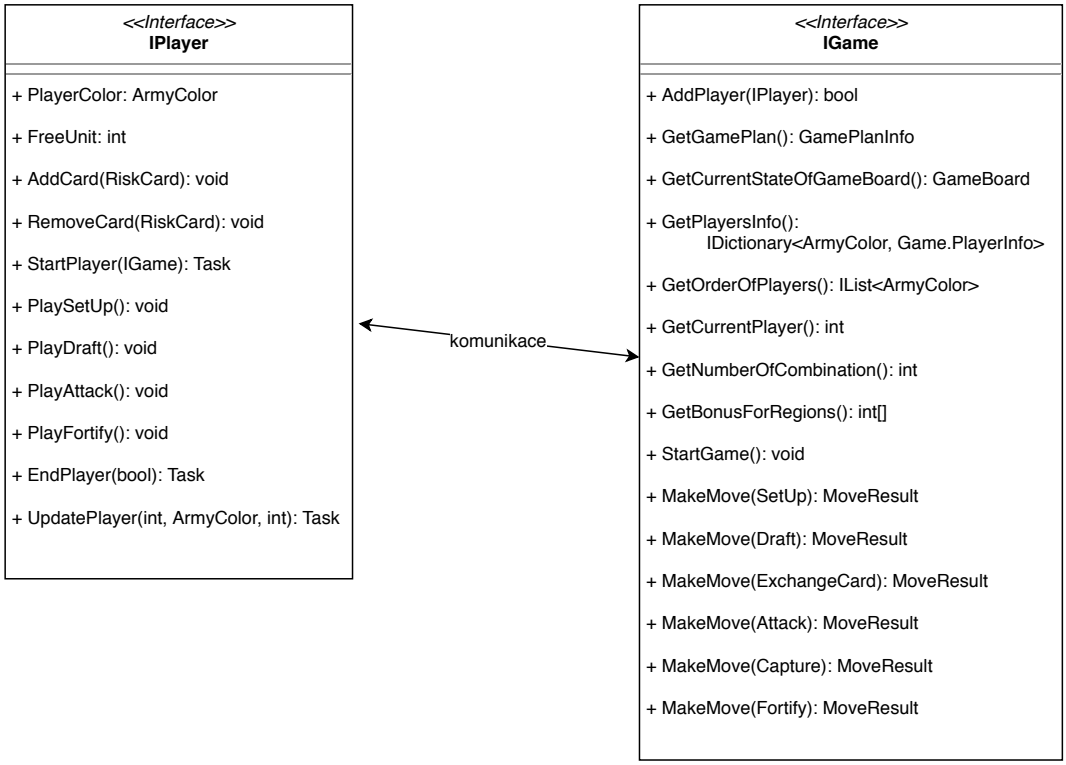
zavolá metoda EndPlayer(bool isWinner), která předá informaci o vítězství hráče a ukončí hru. Celý proces komunikace je zachycen stavovým diagramem na Obrázku 3.1.



Obrázek 3.1: Stavový diagram

Kromě provádění tahů musí hra posílat aktualizace o stavu hry. Potřebujeme získávat nové armády, karty a změny stavu území. Pro stav armád má interface hráče celočíselnou property FreeUnit, přes kterou hra přiděluje armády na začátku kola nebo po výměně karet. Kartu hra předává hráči pomocí metody AddCard(RiskCard card) a při výměně karet ji odebírá pomocí RemoveCard(RiskCard card). Poslední metodou v interfacu hráče je UpdateGame(byte areaID, ArmyColor armyColor, int sizeOfArmy), která umožňuje předat aktualizace hry. Hra rozlišuje jednotlivé hráče podle jejich barvy, proto má ještě interface property PlayerColor. Souhrn interfacu hráče a hry je na Obrázku 3.2.

Interfacy IPlayer a IGame tvoří jádro hry a popisují vzájemnou komunikaci. Pro agenty je připravený ještě jeden interface, který dědí od IPlayer interfacu. Přidává navíc čtyři property pro měření času reakce agenta v jednotlivých fázích hry. Je to interface hlavně pro diagnostiku agenta a pro porovnání agentů. Aby agent mohl hrát, stačí, když naimplementuje interface IPlayer.

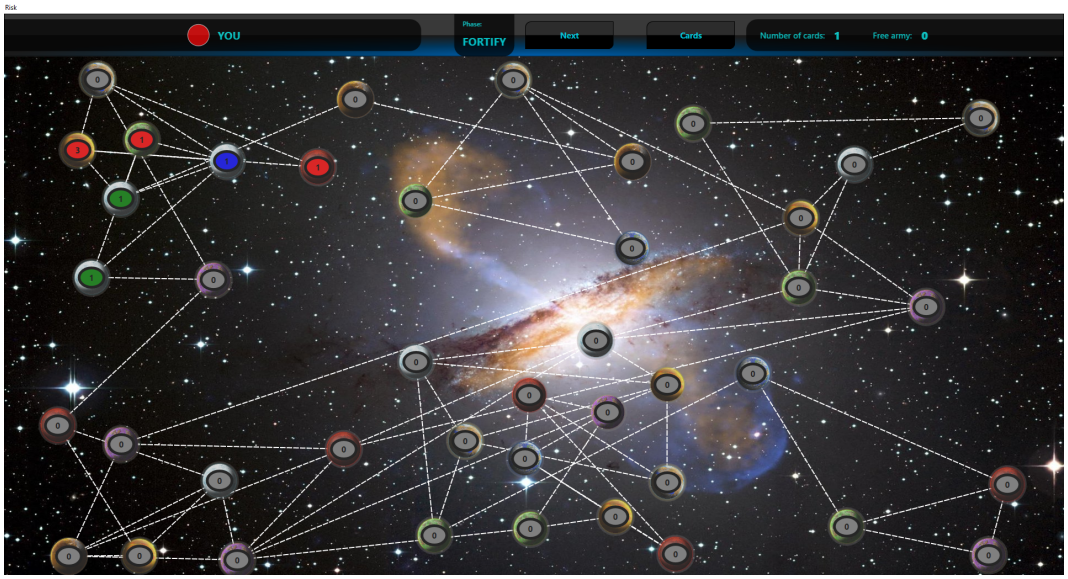


Obrázek 3.2: Rozhraní hry

Interface bez implementace je jen kontrakt, který musí splňovat nějaká třída. Abychom mohli hrát a testovat agenty, potřebujeme nějakou implementaci interfacu IGame. IGame má dvě různé implementace. Původně byla jen jedna hlavní implementace, třída Game. Game je implementováno hlavně pro lidské hráče. Provádí rozsáhlé kontroly tahů, zda se jedná o validní tah, případně zda je vůbec hráč na tahu. Na základě toho provedení tahu není moc rychlé, ale lidský hráč nic nezpozoruje. Když implementaci Game začali využívat agenti, hra trvala příliš dlouho a bylo třeba ji zrychlit. Proto třída Game byla pro agenty využita jen pro kontrolu, zda provádí validní tahy. Díky tomu vznikla třída GameSimulation, v níž byly odstraněny všechny nepotřebné kontroly. Tato třída pouze řídí hru a poskytuje agentům možnost provést tah bez všech kontrol. Víme, že agenti provádí vždy validní tah, takže není nutné provádět zbytečné kontroly. Navíc agent při výběru tahu provádí stejné kontroly, jako by prováděl GameSimulation s kontrolami. Pokud si potřebujeme ověřit správnost tahů, můžeme vždy využit třídu Game.

Kromě implementace hry pro agenty je připravená kompletní implementace singleplayeru a multiplayeru s grafickým uživatelským rozhraním. V singleplayeru si můžeme zahrát proti vytvořeným agentům anebo můžeme vyzvat ostatní hráče v multiplayeru. Multiplayer je client-server aplikace. V menu vybereme multiplayer a pomocí klienta se připojíme k našemu serveru, přihlásíme se pod svojí přezdívkou a pak můžeme hru vytvořit nebo se připojit k existující hře. Při vytváření hry si můžeme zvolit název hry, pro kolik hráčů hra bude, a jestli chceme klasickou mapu anebo náhodně generovanou. Po vytvoření hry nás server přenese do čekací místnosti, kde můžeme vidět, jak se postupně připojují ostatní hráči. Až budeme připraveni, stačí to oznámit pomocí tlačítka. Až budou všichni připraveni, začne se hrát. Pokud se budeme chtít připojit k existující hře, stačí

vybrat jednu z her v seznamu a kliknout na připojit. Opět se dostaneme do čekací místnosti, kde posléze začneme hrát. V případě, že nechceme hrát multiplayer, můžeme v menu vybrat singleplayer. Máme na výběr zvolit pro kolik hráčů hra bude, jakou chceme mapu a proti jakému agentovi si chceme zahrát. Implementace hry je vytvořená ve WPF s vesmírnou tématikou. Na Obrázku 3.3 můžeme vidět jednu ukázku. Podrobnější popis architektury aplikace a uživatelská dokumentace se nachází na CD, jehož obsah je popsán v příloze C.



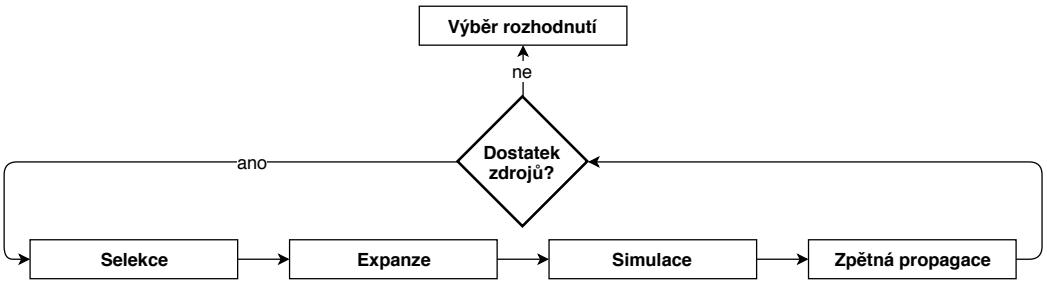
Obrázek 3.3: Ukázka hry

4. Druhy Agentů

4.1 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS)[6] je algoritmus, který umožňuje vybrat nejoptimálnější rozhodnutí na základě současného stavu. Algoritmus se trochu podobá algoritmu Minimax, ale dosahuje lepších výsledků. Stejně jako v Minimaxu i zde se buduje strom, jehož uzly představují stavy a každý uzel má své ohodnocení. Výhodou MCTS je, že se neprohledává celý stavový prostor od současného stavu do určité hloubky stromu, ale jen stavy, které nám dávají určitou jistotu úspěchu. Na to, co je myšleno jistotou, se podíváme dále v této kapitole. Tohoto efektu můžeme docílit i v Minimaxu, když ho rozšíříme o $\alpha - \beta$ ořezávání. Problémem je znalost domény. Aby programátor napsal dobrý Minimax a $\alpha - \beta$ ořezávání, musí mít už nějakou netriviální znalost problému. Především musí vědět, jaké ohodnocení přiřadit danému stavu a které stavy už nemá význam prohledávat. Tato znalost pak primárně ovlivňuje kvalitu Minimaxu. MCTS pracuje jinak. Jak už z názvu vyplývá, je založený na metodě Monte Carlo, která pro výpočet využívá náhodné simulace. MCTS provede mnoho stochastických simulací a na základě toho vybere optimální řešení. Díky tomu nemusíme mít skoro žádnou znalost domény a přitom můžeme dosáhnout lepšího výsledku než s Minimaxem.

Základním principem MCTS je postupně rozšiřovat a prohledávat optimální uzly stromu. Až je splněna ukončovací podmínka, vrátí nejoptimálnější rozhodnutí. Algoritmus se skládá ze čtyř fází: selekce, expanze, simulace a zpětná propagace, které probíhají přesně v tomto pořadí. V každé iteraci algoritmu se projdou všechny fáze. Výpočet probíhá, dokud je dostupný určitý typ zdrojů (čas, paměť nebo pevně daný počet iterací). Po ukončení prohledávání proběhne výběr nejoptimálnějšího rozhodnutí a tímto je algoritmus hotov. Průběh algoritmu je znázorněn na Obrázku 4.1, následující kapitoly jsou pak zaměřeny na podrobnější popis jednotlivých fází.



Obrázek 4.1: Průběh algoritmu

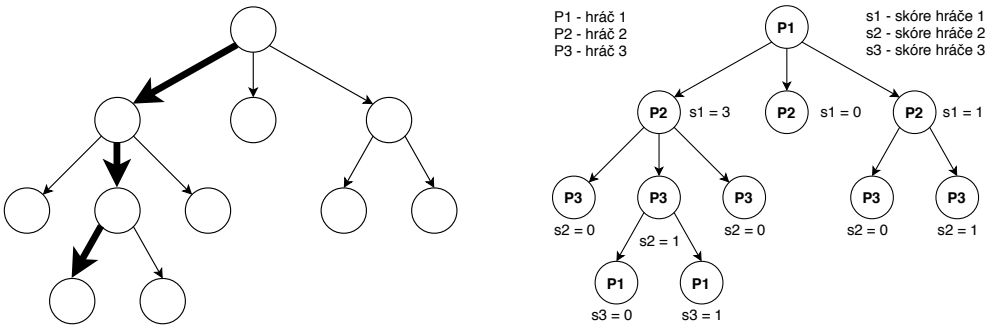
4.1.1 Selekce

Cílem selekce je vybrat nejoptimálnější uzel, který nám dává největší jistotu úspěchu. Začíná se v kořeni stromu a pokračuje se dále až k listu stromu za pomoci strategie selekce. Jedna z nejpopulárnějších strategií je UCT (Upper Confidence Bound), která byla využita i k implementaci MCTS v AlphaGo. Nyní se podíváme

blíže na naši jistotu úspěchu, kterou nám vyjadřuje UCT funkce:

$$UCT(v_i, v_p) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\log(N(v_p))}{N(v_i)}}$$

Parametry funkce tvoří v_i i-tý potomek otce v_p. Funkce $Q : V \to N^+$ určuje počet odměn daného uzlu $v \in V$ a funkce $N : V \to N^+$ určuje, kolikrát byl daný uzel $v \in V$ navštíven, pak na první zlomek $\frac{Q(v_i)}{N(v_i)}$ se můžeme dívat jako na poměr odměn ku kolikrát byl uzel navštíven. To je určitě dobrá hodnota. Chceme vybrat uzly, které jsou nejúspěšnější a obdržely nejvíce odměn. Někdo by si mohl říct, že tato hodnota nám stačí, ale zahodila by nám rozhodnutí, která ze začátku nedopadla dobře, nedostala odměnu, ale v dalším průběhu výpočtu by se mohla jevit jako lepší volba. Proto má UCT další část odmocninu, která nám umožňuje prozkoumat uzly, které zrovna nedopadly dobře, ale stále je tam možnost, že by se mohly vylepšit. Poslední, co nám zbylo, je konstanta c, která se volí okolo hodnoty $\sqrt{2}$. V případě implementace MCTS pro nějakou hru funkce Q přiřazuje odměny uzlu pro hráče, který provedl tah v otci a tím se dostal do daného uzlu. Výpočet UCT zůstává stejný. Pro daného hráče se vždy snažíme vybrat ten nejoptimálnější tah.



(a) Selekce (b) Odměny hráčů

V Alpha GO Lee a Alpha Zero provádí selekci úplně stejně, jen s vylepšenou UCT funkcí, která ještě lépe vybírá nejoptimálnější uzly a více zmenšuje stavový prostor. Funkce, kterou vyvinuli, je následující

$$UCT(v_i, v_p) = \frac{Q(v_i)}{N(v_i)} + c\,\mathsf{P}(v_p, v_i)\sqrt{\frac{N(v_p)}{1+N(v_i)}}$$

První část UCT je stejná, ale přibyla nová hodnota $\mathsf{P}(v_p, v_i)$, která určuje pravděpodobnost tahu, který nás dostane z otce do syna. Pravděpodobnost je získávána z deep neural network, která obdrží stav hry a vydá pravděpodobnost tahu. Tímto vylepšením MCTS dosahuje ještě lepších výsledků. Selekce je velice důležitá, protože nám vybírá nejoptimálnější uzly a tím nemusíme prohledávat řadu úplně špatných uzlů. Špatná strategie selekce nám může vybírat špatné uzly a my budeme muset prohledat více možností, případně může dojít k situaci, kdy špatná strategie selekce zahodí i dobré uzly.

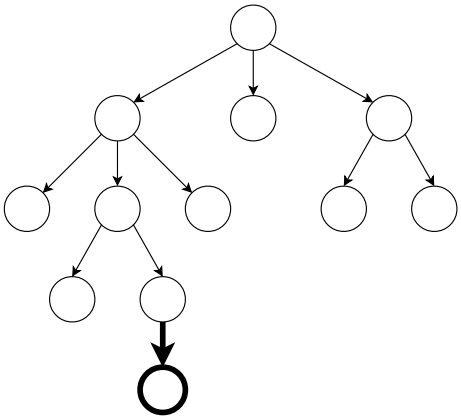
4.1.2 Expanze

Definice 5. *Navštívený uzel označíme takový, u kterého proběhla fáze simulace.*

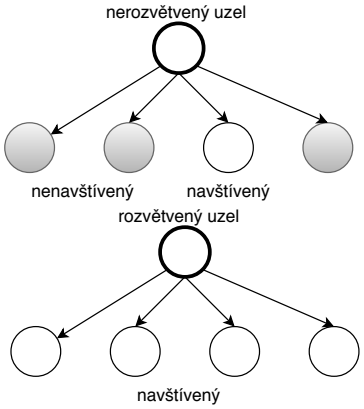
Definice 6. *Plně rozvětvený uzel označíme takový, jehož všichni potomci byli navštíveni a nelze už přidat dalšího potomka.*

Definice 7. *Nerozvětvený uzel označíme takový, jehož alespoň jeden potomek nebyl navštíven nebo lze přidat dalšího potomka.*

Cílem expanze uzlu je vydat možný následující stav nebo stavy. Všechny uzly stromu můžeme rozdělit na dva typy: plně rozvětvený uzel a nerozvětvený uzel. U plně rozvětveného uzlu už nemáme jak dál expandovat a musíme vybrat jednoho z potomků. U nerozvětveného uzlu máme dva přístupy, jak postupovat. Pokud máme dostatek paměti, může expanze probíhat tak, že přidáme všechny možné následující stavy. Pokud máme pouze omezenou paměť nebo stavy zaberou příliš mnoho paměti, můžeme přidat pouze jeden následující stav. Až opět vybereme tento uzel můžeme přidat další následující stav, pokud existuje.



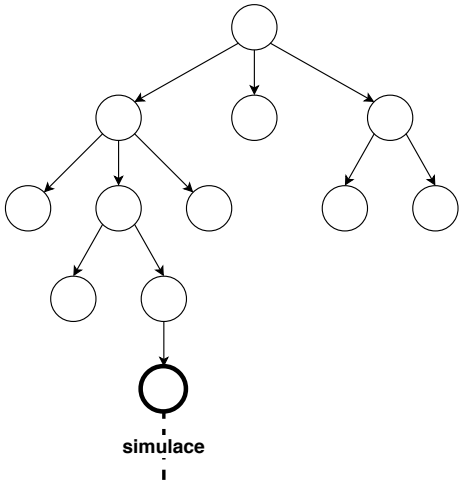
(a) Expanze



(b) Nerozvětvený a rozvětvený uzel

4.1.3 Simulace

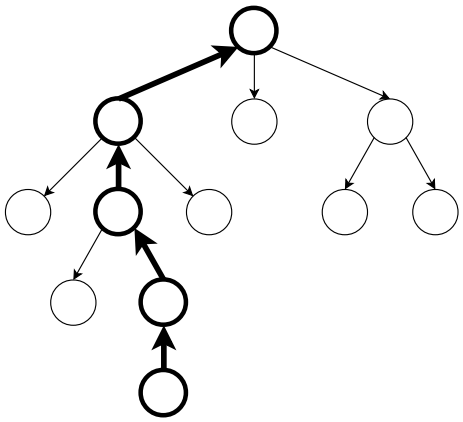
Cílem simulace je odsimulovat danou situaci až do jejího konce nebo do situace, ze které lze vyvodit nějaký výsledek. Simuluje se vždy stav nově přidaného uzlu. Důležité je, aby simulace proběhla velice rychle, protože chceme provádět mnoho simulací v jednom výpočtu MCTS. Existují dvě strategie, jak provádět simulaci. Můžeme využít náhodnou simulaci, kdy každé následující rozhodnutí je vybráno s pravděpodobností p. Výhodou této strategie je, že nemusíme při výběru rozhodnutí provádět žádný složitý výpočet a při simulaci jsou vzaty v potaz opravdu všechna možná rozhodnutí. Nevýhodou může být fakt, že strategie nemusí rychle konvergovat k nějakému výsledku, a proto potrvá příliš dlouho. Oproti tomu můžeme zvolit strategii, která vybírá následující rozhodnutí chytřeji. Můžeme přidat nějakou rozumnou heuristiku. Rozumnou, protože si nemůžeme dovolit, aby se heuristika počítala příliš dlouho. Výhodou strategie s heuristikou je možná rychlejší konvergence k výsledku, kvalitnější rozhodnutí, oproti tomu nevýhodou může být příliš dlouhý výpočet heuristiky. Po provedení simulace vyhodnotíme výsledek a předáme ho fázi zpětné propagace. Při simulaci nevznikají žádné nové uzly stromu.



Obrázek 4.4: Simulace

4.1.4 Zpětná propagace

Cílem zpětné propagace je rozšířit výsledek simulace do svých prarodičů a zvýšit počet celkových navštívení uzlů. Zpětná propagace prochází cestou, kterou jsme se dostali do daného uzlu. Výsledkem simulace rozumíme přiřazenou odměnu. V případě her můžeme mít odměny typu: 1 za výhru, $\frac{1}{2}$ za remízu a 0 za prohru. U více jak dvou hráčů odměna za remízu je obecně $\frac{1}{n}$, kde n je počet hráčů. Kromě informace o výši odměny musíme šířit, i komu odměna patří. Odměny musíme přiřazovat ke správným uzlům, kde daný hráč odehrál tah.



Obrázek 4.5: Zpětná propagace

4.1.5 Výběr rozhodnutí

Posledním úkolem MCTS je vybrat nejoptimálnější rozhodnutí. Vybíráme vždy z potomků kořene. Když se podíváme na průběh selekce a na to, jak je stavový prostor prohledáván, tak nejoptimálnějším potomkem kořene bude uzel s největším počtem navštívení. Tento uzel byl mnohokrát odsimulovaný a získal nejvíce odměn, takže máme velkou jistotu úspěchu. Po výběru rozhodnutí celý algoritmus končí.

4.1.6 Pseudo kód

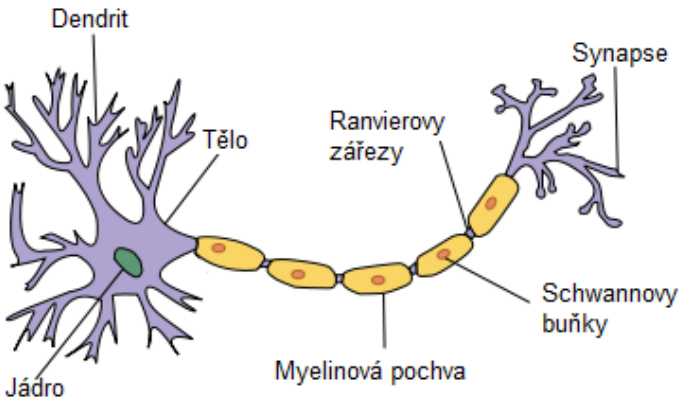
```
function Select(root) returns best node
begin
   node = root
   while node is fully expanded do
       begin
           node = BestUCTChild(node)
       end
   if node is fully expanded then
       begin
           return node
       end
   else
       begin
           return ExpandNode(node)
       end
end

function BestUCTChild(node) returns child with highest UCT score
begin
   return child with highest UCT score
end

function ExpandNode(node) returns new child
begin
   return create new random child
end

function Simulate(node) returns result of simulation
begin
   return simulate state of node
end

function BackPropagate(node, result)
begin
   while node is not root do
       update rewards and visits of node
end

function MonteCarloTreeSearch(root) returns best decision
begin
   while resources left do
       begin
           bestNode = Select(root)
           result = Simulate(bestNode)
           BackPropagate(bestNode, result)
       end
   return child with highest number of visits
end
```

4.2 Neuronová síť

V dnešní době jsou počítače schopny provádět složité výpočty během mrknutí oka, ale stále jsou zde problémy, které neumí tak snadno řešit. I když se mozek nemůže rovnat s výpočetní silou počítače, pořád má jednu vlastnost, kterou počítač nemá. Mozek se dokáže učit. Od narození se člověk naučí chodit, mluvit, rozpoznávat věci okolo, pak ve škole rozvine své další znalosti, ale tím to nekončí a učí se až do konce života. Díky učení se člověk naučí řešit problémy, které dosud řešit neuměl. Učení však vyžaduje hodně času. Navíc se člověk nedokáže učit celých 24 hodin denně. Počítač se naproti tomu neunaví a kdyby se uměl učit, dokázal by se naučit řešit věci, které by člověku zabraly mnoho času. Pokud by vůbec byl člověk schopný se naučit daný problém řešit. Myšlenka učení a princip fungování mozku vedl ke vzniku neuronových sítí[7].

4.2.1 Biologický základ

Mozek[8] spolu s míchou tvoří centrální nervovou soustavu, která je hlavní řídící jednotkou organismu. V nervovém systému se ještě nachází periferní nervová soustava, která propojuje celé tělo. Hlavní částí nervové soustavy je neuron. Neuron se propojuje s dalšími neurony a spolu tvoří obrovskou neuronovou síť. Díky této síti jsme schopni přijímat informace z okolí, umožňuje řízení organismu, vznik myšlenek apod. Jen mozek obsahuje přibližně 100 miliard různě propojených neuronů. Právě neuron rozhoduje o tom, kam informace poputuje a jak se zpracuje.



Obrázek 4.6: Neuron[9]

Neuron se skládá z několika částí. Pomocí dendritů, které tvoří výběžky těla neuronu, přijímá informace ve formě elektrických signálů od ostatních neuronů. Pokud je signál dostatečně silný, vyvolá nervový vzruch a informace projde tělem neuronu do jádra, které je uchováno v těle. Jádro přijme informaci, zmoduluje ji a přepošle dále přes axon. Některé neurony nemodulují informaci a jen ji přeposílají dále. Axon propojuje tělo neuronu se synapsemi a přenáší informace. Pro rychlejší přenos informací může být axon obalen myelinovou pochvou, která přenos urychlí. Nakonec informace doputuje do synapsí, které jsou propojené s dendrity dalších neuronů. Tady o přenosu rozhodnou neurotransmitery. Neurotransmitery jsou chemikálie, díky jejichž množství a typu se informace buď přenese nebo nepřenese. Pokud se budeme učit něco nového, začne se nám budovat určitý okruh

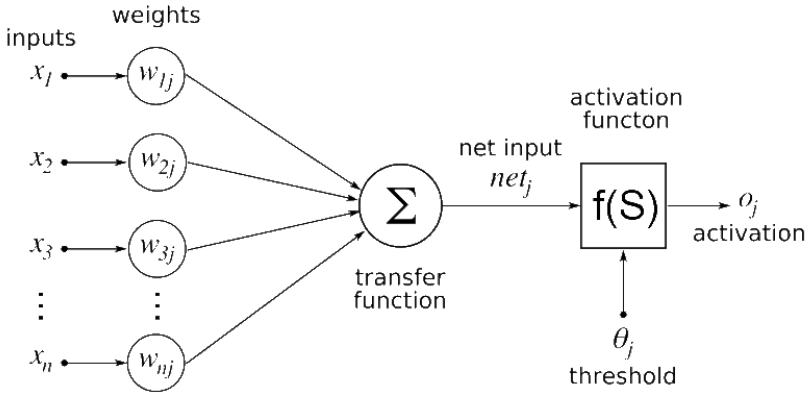
spojení, který postupně bude sílit. Proto nám učení trvá nějakou dobu. Naopak pokud přestaneme využívat nějakou informaci nebo provádět nějakou činnost, okruh spojení začne slábnout.

4.2.2 Matematický Model Neuronové Sítě

Matematický model neuronu je podobný výše zmíněnému biologickému neuronu. V textu dále budeme myslet neuronem matematický model neuronu na Obrázku 4.7. Každý neuron má $x_1 \dots x_n$ vstupů, které si můžeme představit jako dendrity. To jestli se vyvolá nervový vzruch a jak silný bude, nám určuje váha každého vstupu $w_1 \dots w_n$. Váhy nám nahrazují princip neurotransmiterů. Výsledný i-tý vstup je pak $x_i w_i$, kdy hodnota vstupu se přenásobí váhou vstupu. Než se vstupy dostanou do jádra neuronu, všechny se sečtou. Na součet vstupů se v jádře aplikuje aktivační funkce a výsledná hodnota se předá na výstup neuronu. Výpočet neuronu lze vyjádřit jako:

$$out = f(\sum_{i=1}^{n} x_i w_i)$$

Neuron[10] samotný dokáže řešit velmi jednoduché problémy jako například spočítat AND funkci, ale pokud bychom chtěli vyřešit funkci XOR, tak už se nám to nepovede. To neznamená, že neuron je špatný. Vzpomeňme si, že mozek má přibližně 100 miliard různě propojených neuronů. Proto schopnost řešit složité problémy se ukazuje až po propojení více neuronů dohromady. Systém neuronů se nazývá umělá neuronová síť. Neurony se propojují pomocí svých vstupů a výstupů. Každý výstup neuronu je buď finálním výstupem celé neuronové sítě nebo tvoří vstup $1 \dots n$ neuronů. Spojení neuronů budeme nazývat synapse. Máme dva druhy umělých neuronových sítí. První je dopředná neuronová síť (feed-forward network), jejíž výstupy neuronů jdou pouze jedním směrem. Žádný výstup se nevrací do neuronu, kde už výpočet proběhl. Neuronová síť tvoří orientovaný acyklický graf. Druhá je rekurentní neuronová síť (recurrent network), jejíž některé výstupy neuronů se vrací zpět jako vstupy neouronů. Tato vlastnost umožňuje, že síť může mít svůj interní stav, který může záviset na předchozím vstupu. Výpočet neuronové sítě je více dynamický až chaotický. Z tohoto hlediska se jeví jako zajímavější model, než je dopředná neuronová síť, ale o to komplikovanější. My se budeme zabývat prvním modelem, který poté využijeme k implementaci agenta.

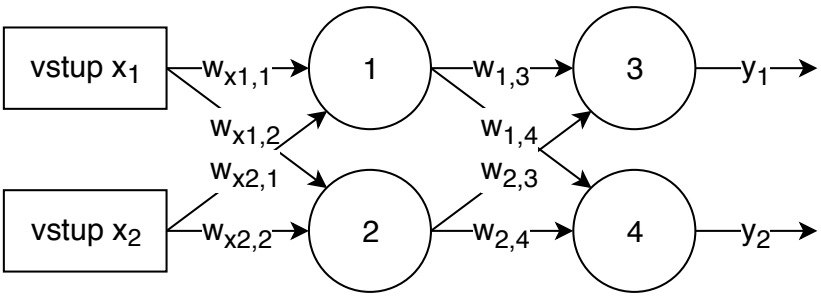


Obrázek 4.7: Model Neuronu [11]

Dopředná neuronová síť se obvykle rozděluje do vrstev. Dva neurony jsou ve stejné vrstvě, pokud mezi nimi nevede cesta ze synapsí. Na základě toho máme jednovrstvé neuronové sítě, které mají jenom jednu vrstvu neuronů přímo propojenou se vstupem a výstupem, a vícevrstvou neuronovou síť. Vícevrstvá neuronová síť se skládá z jedné a více skrytých vrstev a jedné výstupní vrstvy. Zkusme vyřešit náš problém s funkcí XOR pomocí jednovrstvé neuronové sítě. Pokud bychom vytvořili neuronovou síť s dvěma neurony v jedné vrstvě a zkusili ji naučit funkci XOR, opět bychom selhali. Problémem je, že funkce XOR není lineárně separovatelná a proto jedna vrstva neuronů se ji není schopna naučit. Proto vznikly rekuretní a vícevrstvé neuronové sítě.

Doposud víme, jak probíhá výpočet jednoho neuronu, ale pojďme se podívat, jak vypadá výpočet v celé neuronové síti. Pokud máme jednovrstvou neuronovou síť, výpočet je jednoduchý. Na vstupu dostaneme vektor $(x_1, x_2 \dots x_n)$ a výstupem bude opět vektor o stejné velikosti, protože neurony jsou přímo propojeny se vstupem a výstupem. Výsledný vektor získáme výpočtem každého neuronu zvlášť dle výše zmíněného vzorce. Nyní si představme neuronovou síť, kterou tvoří dvě vrstvy s dvěma neurony jako na Obrázku 4.8. Neuronová síť bere na vstupu vektor (x_1, x_2) a na výstup vydá vektor (y_1, y_2). Má jednu skrytou vrstvu a jednu výstupní vrstvu. Ukažme si, jak se vypočítá výstup y_1. Stejným postupem vypočítáme i výstup y_2 nebo výstupy složitějších neuronových sítí.

$$\begin{aligned} y_1 &= f(w_{1,3}out_1 + w_{2,3}out_2) \\ &= f(w_{1,3}f(w_{x_1,1}x_1 + w_{x_2,1}x_2) + w_{2,3}f(w_{x_1,2}x_1 + w_{x_2,2}x_2)). \end{aligned} \tag{4.1}$$



Obrázek 4.8: Neuronová síť

Otázkou zůstává, jak zvolit počet vrstev a počet neuronů ve vrstvách pro daný problém nebo-li jakou topologii zvolit. Bohužel neexistuje přesný postup, který by vygeneroval pro určitý problém danou topologii. Viděli jsme však, že pro určité problémy je třeba zvolit složitější topologii, zatímco u některých stačí pouze jedna vrstva neuronů. Při volbě topologie neuronové sítě tedy musíme experimentovat. Přibližná topologie se dá odhadnout z podstaty problému, ale musí se vyzkoušet, která topologie je schopná se daný problém naučit.

4.2.3 Aktivační funkce

Každý neuron má uvnitř svou aktivační funkci, která definuje výstup. Existuje několik různých funkcí, které se hodí na určité typy problémů. Při správné volbě funkce se bude neuronová síť lépe učit a dosáhneme lepších výsledků. Prvním modelem byl perceptron, který byl využit v roce 1957 k rozpoznání obrázků. Funkce

perceptronu je jednoduchá. Zobrazuje doménu reálných čísel na binární hodnoty 0,1. Oproti tomu můžeme využít plynulejší funkci Sigmoidu, která zobrazuje doménu reálných čísel do otevřeného intervalu (0,1). Sigmoida blíže reprezentuje biologické chování neuronů. Hlavní využití si funkce našla v lineární regresi, ale dále se používá na výstupu v neuronových sítích pro výpočet pravděpodobnosti. Pro výpočet funkce stačí vypočítat:

$$y = \frac{1}{1+\exp(-x)}$$

I když je Sigmoida dobrá funkce, s některými problémy si neumí poradit. Proto se používá obdobná funkce Hyperbolický tangent, který mapuje doménu reálných čísel na otevřený interval $(-1,1)$. Využívá se na podobné problémy jako Sigmoida, ale v případech, kdy neuronová síť má záporné výstupy. Poslední funkci, kterou je dobré zmínit, je tzv. Rectifier funkce. Je to jedna z nejpopulárnějších funkcí, která se využívá na všechny druhy problémů. V učení neuronových sítí dosahuje nejlepších výsledků a přitom ji lze spočítat velice snadno $max(0, x)$. Všechny zmíněné funkce jsou znázorněny v grafu v příloze A.1.

4.2.4 Učení

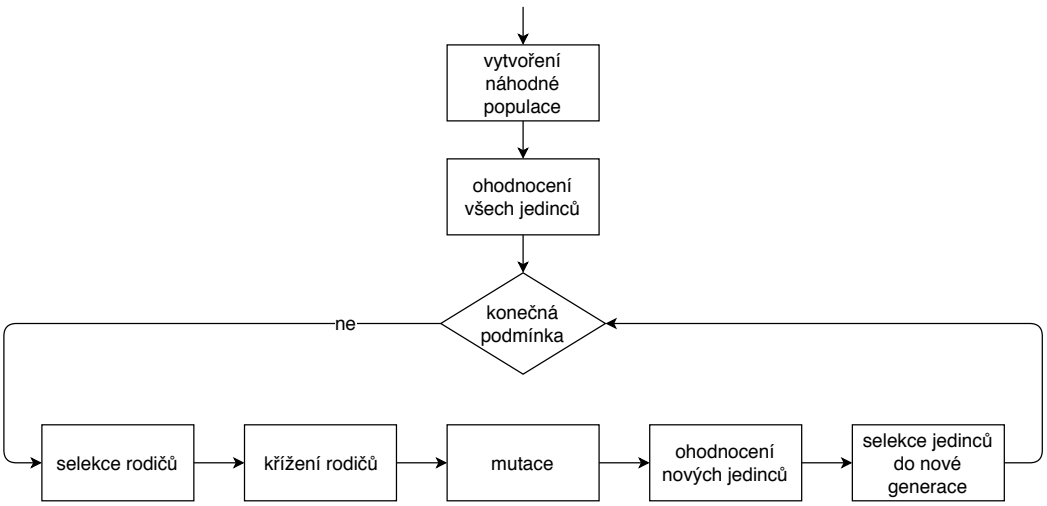
Na začátku povídání o neuronových sítí jsme lehce naznačili, že mají schopnost se učit. V průběhu jejich popisu fungování jsme dokonce mluvili přímo o učení neuronových sítí. Proto ta analogie neuronových sítí a mozku. Schopnost učit se je to nejdůležitější, co dělá neuronovou síť tak skvělým nástrojem. Pod pojmem učení se myslí, že neuronová síť je schopna řešit daný problém lépe a lépe. Proces opakovaného provádění určité činnosti s postupným zdokonalováním se, je učení. Obdobně se učí i neuronová síť. V průběhu učení neuronová síť opakovaně provádí určitou činnost, pro kterou byla stvořena, a snaží se zdokonalit své řešení. Víme, že výpočet neuronové sítě ovlivňují váhy synapsí. Aktivační funkce také ovlivňují výpočet, ale to už je spíše problém návrhu neuronové sítě. Hlavní neznámou jsou váhy. Změněním vah se buď přiblížíme správnému výsledku nebo vzdálíme. Tedy učení neuronových sítí spočívá v nalezení správných vah synapsí.

Přístupy k učení rozlišujeme na 3 typy: supervised learning, unsupervised learning a reinforcement learning. Supervised learning jsou metody učení, kdy neuronové síti poskytujeme jak množinu vstupů, tak i množinu výstupů. Známe-li správný výstup neuronové sítě, jsme schopni porovnat současný výstup se správným výstupem a spočítat chybu výstupu. Chyba nám určuje jak daleko jsme od správného výstupu. Z chyby se dokážeme poučit. Vezmeme ji a budeme ji zpětně postupně propagovat skrze všechny neurony a v každém neuronu se pak na základě chyby budeme snažit vylepšit výsledek. Jak už napovídá postup učení, jedna z metod supervised learningu je zpětná propagace, která využívá výpočtu gradientu k úpravě vah synapsí. Unsupervised learning jsou metody učení, kdy máme k dispozici pouze množinu vstupů. Místo abychom pro daný vstup hledali správný výstup, snažíme se, aby neuronová síť našla mezi množinou vstupů nějaký vzorec. Supervised a unsupervised learning se nejčastěji využívá v data miningu. Poslední, a pro nás nejdůležitější, je reinforcement learning. Reinforcement learning je podobný supervised learningu. K dispozici máme množinu vstupů, ale místo množiny výstupů máme systém odměn. Odměna je ohodnocení výstupu, jak moc je dobrý. Principem reinforcement learningu je snažit se zvětšovat zisk

odměny. Čím vyšší odměna, tím lepší výstup neuronová síť vydala. Jednou z metod reinforcement learningu jsou například evoluční algoritmy, o kterých budeme mluvit v následující kapitole. Naším hlavním zájmem bude reinforcement learning, protože se ideálně hodí pro náš charakter problému. Z tohoto důvodu bude reinforcement learningu věnována ještě další pozornost.

4.3 Evoluční algoritmy

Určitě všichni známe jméno Charles Darwin a jeho evoluční teorii, která spojuje postupnou evoluci druhu s přirozeným výběrem. Evoluční algoritmy, jak už název napovídá, nejsou daleko od Darwinovy teorie. Evolučními algoritmy je skupina algoritmů, která je založena na myšlence evoluce. Jednotlivé algoritmy se liší pouze v malých detailech, ale průběh mají stejný. Stejně jako v přírodě máme množinu jedinců, která se postupem času a okolním vlivem vyvíjí, i v evolučních algoritmech pracujeme s množinou jedinců. Jedincem budeme chápat jedno z možných řešení, které vůbec nemusí být správné ani optimální. Množinu jedinců budeme označovat populace. Přesněji populace je multimnožina jedinců, protože může obsahovat dva stejné jedince. Další specifickou množinou jedinců je generace, která seskupuje jedince vzniklé ve stejný čas. Generaci určuje počet proběhlých křížení a mutací, k nimž došlo předtím, než vznikl daný jedinec. Křížení a mutace je proces, kdy vzniknou noví jedinci z předchozích jedinců. Nultou generací budeme označovat množinu náhodně stvořených jedinců, kdy neproběhla žádná mutace ani křížení. Pokud na této nulté generaci provedeme křížení nebo mutaci, získáme první generaci. Populace může obsahovat jedince z různých generací. Poslední, co nám zbývá v evoluci, je přirozený výběr. K tomu budeme potřebovat funkci, která specifikuje, jak je daný jedinec kvalitní. Takovou funkci budeme označovat jako fitness funkce. Fitness funkce nám ohodnotí všechny potřebné jedince a pak můžeme provést výběr. Celý průběh algoritmu je znázorněný na Obrázku 4.9. [12]



Obrázek 4.9: Evoluční algoritmus

4.3.1 Populace

Abychom mohli vytvořit populaci, musíme si nadefinovat, jak budou vypadat jedinci. Jedinec reprezentuje jedno možné řešení problému, ale otázka je, jak bude reprezentace vypadat. V tomto místě se evoluční algoritmy rozcházejí. Strukturu jedince budeme nazývat genotyp. Genotyp je obvykle tvořen datovou strukturou. V genetických algortimech (Genetic Algorithms) genotyp tvoří řetězec nad konečnou abecedou. Další možností je vektor reálných čísel, který se využívá v evolučních strategiích (Evolution Strategies). Datové struktury mohou být ale složitější. Genotypem může být konečný stavový automat, který je využíván v evolučním programování (Evolutionary Programming), nebo datová struktura strom v genetickém programování (Genetic Programming). Samotný genotyp je pouze nosičem informace. Důležité je nadefinovat význam informace. Význam informace budeme označovat jako fenotyp. Například při evoluci neuronových sítí bychom mohli použit genotyp vektor reálných čísel a fenotyp by nám říkal, že vektor reálných čísel představuje váhy synapsí. Nyní máme hotovou strukturu jedince, která reprezentuje jedno možné řešení daného problému. Jedinec je také někdy označován jako chromozom. Pro vytvoření populace potřebujeme znát její velikost. Velikost populace se skoro ve všech algoritmech v průběhu evoluce nemění a je konstantní. Jedinec se také nemění. Abychom získali jiného jedince, musíme provést v populaci křížení nebo mutaci, ale selekce nám pořád zachová stejnou velikost populace. Vhodní jedinci přežijí další generaci a zůstanou v populaci a ostatní vypadnou.

4.3.2 Fitness Funkce

Fitness funkce je hlavním elementem při selekci. Funkce přiřazuje ohodnocení jedinci. Čím vyšší ohodnocení, tím je jedinec lepší. Ohodnocení nám říká, jak je jedinec kvalitní. Slovem kvalitní je myšleno, jak dobře řeší daný problém. Funkce musí být dobře navržená, protože na základě ní probíhá evoluce. Pokud bude funkce přiřazovat nesprávné ohodnocení, bude evoluce směřovat špatným směrem a nepovede k řešení. Například v evoluci neuronové sítě na výpočet funkce AND můžeme ve fitness funkci spočítat vzdálenost výsledků od správných výsledků. Pro zjednodušení řekněme, že funkce bude brát v parametru rovnou neuronovou síť, ale ve skutečnosti by to byl vektor reálných čísel. Potřebujeme, aby se přiblížením ke správnému výsledku zvyšovalo ohodnocení fitness. To nám může zaručit například fitness funkce níže.

$$fitness(x) = 1 - x(0,0) + 1 - x(0,1) + 1 - x(1,0) + x(1,1)$$

4.3.3 Selekce

V celém procesu evoluce máme dva druhy selekcí. Jednou z nich je selekce rodičů. Tato selekce se využívá při křížení nebo mutaci, kdy je potřeba vybrat vhodné rodiče. Výběr probíhá na základě pravděpodobností. Čím kvalitnější jedinec, tím větší má pravděpodobnost být vybrán. Všichni jedinci mají alespoň minimální pravděpodobnost být vybráni. Samotná selekce rodičů spolu s křížením a mutací by nám pouze rozšiřovala populaci. Proto musíme využít další selekci, která nám řekne, kdo z jedinců přežije. Jednou z možností je vybírat pouze nově vzniklé potomky. Evoluce u těchto potomků probíhá díky selekci rodičů. Druhou

z možností je vybírat jedince z celé populace. V tomto případě se využívá selekce elity, která většinou vybírá nejkvalitnější jedince na základě fitness funkce, ale do rozhodnutí výběru může být započítána i generace jedince. Tyto selekce nám zaručují evoluci jedinců.

4.3.4 Křížení

Křížení je obvykle binární funkce, která vezme dva jedince jako rodiče, zkombinuje je a vytvoří jednoho nebo více nových jedinců. Některé evoluční algoritmy využívají křížení s vyšší aritou. Takové křížení už nemá moc biologické základy, i když někdy může dosahovat lepších výsledků. Máme několik způsobů, jak zkombinovat dva jedince. První z nich je jednobodové křížení. V jedinci se vybere náhodně jeden bod, který jedince rozdělí na dvě části. Tyto dvě části si rodiče prohodí a vzniknou tak dva různí jedinci lišící se od rodičů v jedné části. Obdobně funguje multibodové křížení, kdy je jedinec rozdělen na dvě a více částí. Oproti bodovým křížením existuje uniformní křížení, kdy místo rozdělování na části, se prohazují jednotlivé geny. Křížení nezaručuje, že vznikne lepší jedinec. Může vzniknout stejně dobrý, ale i horší. To, jestli dojde ke křížení, nám udává pravděpodobnost křížení, která obvykle bývá vyšší než pravděpodobnost mutace.

4.3.5 Mutace

Mutace je unární funkce, která pracuje s genotypem jednoho jedince. Vezme genotyp, pozmění ho a tím vytvoří nového jedince. Mutace nám udržuje rozmanitost populace a zabraňuje abychom spadli do lokálního extrému. Díky tomu lépe prohledáme prostor řešení. Mutací máme několik druhů. Pokud máme genotyp, řetězec binárních hodnot, mutace může probíhat tak, že náhodně vybereme jeden gen a prohodíme binární hodnotu. Dalším možným přístupem aplikovatelným na libovolný genotyp je prohazování. Zvolíme náhodně dva geny a prohodíme je. Můžeme také pracovat s množinou genů. Náhodně zvolíme množinu genů a promícháme je mezi sebou nebo můžeme vzít jejich inverz. Mutaci aplikujeme s určitou pravděpodobností, která bývá nižší. Vyšší pravděpodobnost by zapříčinila náhodné prohledávání.

5. Implementace Risk Agentů

V této kapitole aplikujeme teorii agentů a vytvoříme 4 agenty. V první řadě se podíváme na agenta, který bude provádět náhodné tahy. Následně tohoto agenta využijeme při implementaci agenta s MCTS. U třetího agenta se pokusíme aplikovat neuronové sítě, které využijeme také jako heuristiku u agenta s MCTS.

5.1 Náhodný agent

Jako úplně první agent bude ten nejjednodušší a nejhloupější. Bude to náhodný agent. I když to nebude moc chytrý agent, bude se nám hodit hned ve dvou případech. Za prvé bude dobrým nástrojem pro otestování rychlosti hry. Za druhé bude dobrým počátečním bodem pro otestování inteligence agenta. Pokud nějaký agent neporazí náhodného agenta, tak jej nelze považovat za alespoň trochu inteligentního. A v poslední řadě se nám bude hodit pro implementaci MCTS agenta. Abychom mohli provádět simulaci v MCTS potřebujeme agenta, který nám rychle odehraje hru až do konce. Zvolením strategie náhodného agenta, který bude provádět náhodné, ale korektní tahy, nemusíme mít nějakou heuristiku. Heuristika v hře Risk může být poměrně složitá. Navíc chceme udělat MCTS, které bude schopné hrát na jakékoliv mapě, takže nemůžeme aplikovat pravidla jako je: zaútoč na Severní Ameriku, první obsaď Austrálii atd. Na základě toho se náhodný agent zdá být vhodnou volbou. Pojďme se nyní podívat na návrh náhodného agenta.

Interface IPlayer nám říká, že agent musí umět odehrát všechny tři fáze kola plus musí umět rozestavit armádu na začátku hry. Při rozestavění armády se náhodný agent podívá na mapu, najde všechny možnosti, kam umístit armádu, se stejnou pravděpodobností vybere jednu z nich a provede tah. V první fázi kola je to trošku komplikovanější. Pokud má více než 5 karet, musí je vyměňovat, dokud počet karet nesníží na méně než 5. Po snížení počtu karet stále může mít nějakou kombinaci karet. Proto agent musí zkontrolovat, jestli v ruce nedrží nějakou kombinaci. Pokud ano, musí náhodně zvolit, zda vyměnit kombinaci karet nebo ne. Následně musí najít všechny možnosti, kam může doplnit armádu. Opět je potřeba náhodně vybrat jednu z možností, ale navíc je potřeba náhodně určit, kolik volné armády doplní. Agent musí doplnit všechnu volnou armádu. Proces náhodného výběru možnosti a doplnění armády musí opakovat tak dlouho, dokud nedoplní všechnu volnou armádu. Poté může přejít do fáze útoku. Útočit může libovolně dlouho, pokud má s čím. Cyklus útočení musí začít rozhodnutím útočit či neútočit. Jestliže se rozhodne útočit, musí náhodně vybrat útočící území a místo, kam zaútočit. Nakonec na základě stavu armády na svém území musí náhodně vybrat velikost útoku a tím dokončit jeden útok. Pokud dobude území, je třeba náhodně zvolit počet armád k přesunu a případně zařadit území mezi možné útočící území. V poslední fázi má možnost přesunout armádu z jednoho území na druhé. Opět se náhodně rozhodne, zda přesunout armádu nebo ne. Jestli bude přesouvat armádu, musí náhodně vybrat dvě svá propojená území, kdy z jednoho přesune armádu na druhé. Počet armád k přesunu je určen náhodně.

Když necháme hrát proti sobě takové agenty, zjistíme, že hra trvá příliš dlouho. V MCTS potřebujeme, aby simulace proběhla velice rychle. Časy reakce agenta

v jednotlivých fázích hry jsou natolik malé, že jsou zanedbatelné a problém musíme hledat jinde. Do času reakce agenta je započítáno i provedení tahu, takže obsluha hry nám to také nebrzdí. Prozkoumáme-li v průběhu hry stav herního plánu, uvidíme, že agenti mají na mapě obrovskou armádu, ale moc nebojují. Problémem je fáze útoku. Navrhli jsme, že možnosti útočení a neútočení budou vybírány náhodně se stejnou váhou. To nám zapříčinilo, že agent je silně defenzivní, ale hra je o dobývání světa, a proto agenti musí chtít dobýt svět. Musíme zvýšit pravděpodobnost útočení. Mějme pravděpodobnost p určující pravděpodobnost útoku a $1 - p$ pravděpodobnost určující neútočení. Zkusme otestovat, která pravděpodobnost p dosáhne lepších výsledků. Necháme náhodné agenty odehrát 500 her pro 3 hráče na standardní mapě a postupně budeme měnit pravděpodobnost p (viz Tabulka 5.1).

p	0,50	0,78	0,79	0,80	0,81	0,82	0,83
čas	∞	1160,1	936,9	916,34	801,0	886,0	981,8
průměrný čas	∞	2,32	1,87	1,83	1,60	1,77	1,96
p	0,84	0,85	0,86	0,87	0,88	0,89	0,90
čas	788,4	835,6	796,9	722,8	752,9	838,1	942,1
průměrný čas	1,58	1,67	1,56	1,45	1,51	1,68	1,88

celkové časy (s) 500 her a průměrný čas jedné hry (s) s pravděpodobností p

Tabulka 5.1: Test pravděpodobností 1

Můžeme vidět, že při vyšší pravděpodobnosti útoku jsou časy kratší a čím je hráč defenzivnější, tím je hra pomalejší, ale příliš vysoká pravděpodobnost opět hru zpomaluje. Časy her jsou stále příliš vysoké, abychom mohli takového hráče využít v MCTS. Při hlubší analýze problému objevíme, že hráči odehrávají příliš mnoho kol. Navzájem si přebírají pár území a nemůžou se dobýt. Dle pravidel při každé výměně karet se následující počet obdržené armády zvýší a díky tomu mají stále mnoho armády. Jedna z modifikací pravidel je omezení obdržené armády za výměnu karet. Zkusme poupravit pravidlo s kartami. Až počet obdržených armád dosáhne 50, přestane se zvyšovat. Po opětovném provedení testů zjistíme, že časy se velice snížily (viz Tabulka 5.2) a můžeme přejít k implementaci MCTS agenta.

p	0,50	0,78	0,79	0,80	0,81	0,82	0,83
čas	∞	54,02	58,57	53,05	37,76	34,38	27,00
průměrný čas	∞	0,11	0,12	0,11	0,08	0,07	0,05
p	0,84	0,85	0,86	0,87	0,88	0,89	0,90
čas	23,63	20,26	19,88	18,81	16,34	15,80	15,76
průměrný čas	0,05	0,04	0,04	0,04	0,03	0,03	0,03

celkové časy (s) 500 her a průměrný čas jedné hry (s) s pravděpodobností p

Tabulka 5.2: Test pravděpodobností 2

5.2 MCTS agent

V kapitole 4 jsme se podívali na to, jak funguje MCTS a nyní ho naimplementujeme pro našeho agenta. V předchozí kapitole jsme si připravili náhodného agenta, kterého využijeme v simulaci stavu v MCTS. Stačí vzít současný stav hry, potřebný počet náhodných agentů a nechat je odehrát hru až do konce. Vítěz simulace hry bude výsledkem simulace, který budeme propagovat nahoru ke kořeni. Hráče budeme označovat číslem, které udává jejich pořadí ve hře. V každém uzlu stromu budeme uchovávat počet výher hráče, který v něm odehrál kolo. Výhry hráče budou reprezentovat odměnu v uzlu. Při selekci budeme využívat klasickou UCT a na konci vybereme uzel s největším počtem navštívení. Výsledkem výpočtu MCTS bude sekvence tahů. Jediné, co nám zbývá, je reprezentace stavu ve stromu a generace nových stavů. Stav ve stromu bude tvořit současný stav herního plánu, číslo hráče, který je na řadě, sekvence tahů, která převedla předchozí stav hry na současný stav, a status hry. Z prvních dvou informací jsme schopni odsimulovat hru. Poslední informace nám uchová výsledek simulace. U statusu budeme potřebovat rozlišit vítěze, neodsimulovanou hru a kořen stromu. Kořen potřebujeme odlišit, abychom v něm neprováděli simulaci. Poslední a největší problém je generování nových stavů.

Z analýzy Risku víme, že větvící faktor je obrovský. Takové velké množství následujících stavů nelze odsimulovat v rozumném čase, když se podíváme na časy náhodného agenta. Navíc bychom ani nemohli prohledávat moc do hloubky. Pokud bychom se podívali na vzniklé stavy, zjistili bychom, že mnoho z nich je stejných. Různé kombinace tahů vedou ke stejnému stavu. Zamezíme-li tomuto efektu, snížíme tím počet nových stavů. K tomu nám poslouží částečné uspořádání. V úvodní fázi hry, kdy rozmísťujeme armádu, můžeme provést pouze jeden tah. Každý možný tah bude reprezentovat jeden možný stav hry a všechny stavy budou různé. Problém nastává v případě fází doplnění armády a útoku. Ve fázi doplnění armády nejprve musíme provést povinné tahy, výměnu karet, abychom snížili jejich počet na povolený. Pokud budeme mít ještě nějakou kombinaci karet, musíme vygenerovat dva případy. Vyměnit karty nebo nevyměnit. Pro každou možnost musíme doplnit na naše území všechny volné armády. Nejprve najdeme všechna území, kam lze doplnit armádu. Každému území přiřadíme index $0 \dots n$ a postupně je všechna zpracujeme následujícím způsobem. Začneme na nultém indexu. Nechť x nám určuje počet volných armád agenta, pak na území můžeme doplnit $0 \dots x$ armád. Pro vygenerování všech možných stavů musíme projít všechny možnosti. Nejprve nedoplníme žádnou, pak jednu armádu a postupně budeme zvyšovat počet až nakonec doplníme všechny armády. Pro každou možnost zvýšíme index a zpracujeme další území. K tomu se nám ideálně hodí rekurze. Na dalším indexu máme opět možnost nedoplnit žádnou armádu nebo postupně zvyšovat počet než doplníme všechny zbylé armády. Jestliže dosáhneme n-tého indexu a stále máme volné armády, doplníme je všechny na území s indexem n a přejdeme do fáze útoku. Pokud nám volné armády dojdou dříve než se dostaneme k poslednímu indexu, opět přejdeme do fáze útoku a už dále nepokračujeme. Částečné uspořádání spočívá v tom, že na zpracované území už nemůžeme doplnit žádnou armádu. Tím získáme jen různé stavy. Ve fázi útoku budeme pracovat podobně. Najdeme všechna území, ze kterých můžeme útočit. Přiřadíme jim indexy $0 \dots n$ a opět je všechna postupně projdeme. Začneme na nultém indexu. Zjistíme možné

cíle a postupně na ně budeme útočit, ale vždy máme možnost neútočit. Nejprve provedeme útoky na první cíl. Dle možností můžeme zaútočit s jednou armádou, dvěma nebo třemi armádami. Pro každý případ získáme nový stav. Pro každou velikost útoku se můžeme opět rozhodnout útočit anebo neútočit a přejít na další cíl. Takhle vygenerujeme všechny stavy po útočení z jednoho území a pro každý stav zvýšíme index a zpracujeme další útočící území. Pokud dobudeme území, musíme ho obsadit. K obsazení potřebujeme přemístit armádu z útočícího území. Minimální počet je velikost útoku a maximální počet je o jedna menší než počet armád na útočícím území. Opět musíme vygenerovat všechny možné stavy. Navíc, pokud obsazené území může útočit, musíme ho přidat na pozici $n + 1$ a také ho zpracovat. Až projdeme všechna území, přejdeme do fáze přemístění armády. Tady nepotřebujeme žádné částečné uspořádání a jen provedeme všechna možná přemístění armád. Pro každý stav získaný z fáze doplnění armády vygenerujeme stavy pro fázi útoku a pro každý z nich vygenerujeme stavy ve fázi přemístění armád. I když jsme využili částečné uspořádání, stále máme velké množství stavů. Přestože jsme v kapitole 4 popisovali, že MCTS nepotřebuje heuristiku, v tomto případě bude zapotřebí.

Potřebujeme vyvinout heuristiku, která nám dostatečně sníží počet vygenerovaných stavů, ale zároveň nám neztratí možné dobré stavy. Úvodní fáze hry nám maximálně vygeneruje 42 stavů, takže tady nebudeme potřebovat heuristiku. Nejvíce stavů nám generují fáze doplnění armády, útok a přemístění armády. Ve fázi doplnění armády nemá význam doplňovat armádu na území neležící na hranici s nepřítelem. S takovou armádou nemůžeme bránit ani útočit. Proto budeme doplňovat armádu jen na území na hranici. Dále nemá úplně význam doplňovat armádu po jedné armádě. Většinou doplníme větší část volné armády tam, kde chceme útočit nebo potřebujeme silnou obranu. Proto při generování stavů budeme přidávat armádu minimálně po dvou armádách. Když bychom chtěli ještě snížit počet stavů, můžeme doplňovat po více armádách. Tato heuristika nám výrazně sníží počet stavů po fázi doplnění armády, ale mnohem více stavů nám generovala fáze útoku. U fáze útoku budeme vycházet z analýzy pravděpodobností kostek a úspěchu útoku. Nemá význam útočit, pokud máme méně armád než obránce. Musíme mít alespoň o dvě armády více. Hlavní rozdíl mezi silami je rozhodující v otázce, zda území dobudeme nebo ne. Proto, když se rozhodneme na nějaké území útočit, budeme útočit vší silou až do konce. Buď ztratíme armádu nebo území dobudeme. Pokud území dobudeme, přesuneme na něj všechnu možnou armádu. Chceme co možná nejvíc armád udržovat na hranicích. Poslední potřebná heuristika je ve fázi přemístění armády. Opět s cílem mít co nejvíce armády na hranicích, budeme přesouvat pouze armádu z území mimo hranice do území na hranicích nebo mezi územími na hranicích. Přesouvat budeme všechnu armádu, protože nemá smysl nechávat armádu za hranicemi a přesouvání mezi hranicemi může podpořit obranu.

Bohužel ani s heuristikou není MCTS schopné zvládat klasickou mapu Risku. Velké množství stavů se nevejde ani do paměti velikosti 8 GB. Problém s pamětí by šel vyřešit zakódováním herního plánu do několika integerů, ale stále nejsme schopni odsimulovat miliony stavů v rozumném čase. Reakční doba MCTS by sahala až k hodinám. Mohli bychom mít menší počet simulací, ale aby měl výpočet MCTS nějakou váhu, musíme stavový prostor prohledat alespoň do nějaké rozumné hloubky. V našem případě se stěží dostaneme do hloubky 2, kdy kořen

je hloubky 0. Maximální počet území, které je MCTS schopné rozumně zvládat, je 18.

Agent využívá MCTS vždy na začátku svého kola. Získá stav hry, předá jej MCTS a spustí výpočet. Po výpočtu MCTS vrátí agentovi sekvenci tahů. V případě úvodu hry sekvence obsahuje pouze jeden tah. Jak jsou na agentovi postupně volány metody na odehrátí určité fáze, agent odehrává příslušné tahy ze sekvence tahů. V případě útoku a přesouvání armád probíhá kontrola, než se provede tah, aby se ověřilo, zda je tah validní. Díky náhodě způsobené hody kostkou nevíme, jaký bude výsledek tahu, a proto je třeba tahy kontrolovat.

5.3 Agent s neuronovou sítí

Než začneme s návrhem neuronové sítě, je potřeba si promyslet, co budeme chtít, aby neuronová síť řešila. V úvodu hry musíme umět dobře rozestavit armádu. V průběhu hry pak máme 3 fáze. V první fázi musíme rozhodnout, zda vyměnit karty nebo ne a kam doplnit armádu. V druhé fázi potřebujeme vědět, kam a jak útočit. Navíc musíme určit počet armád k přesunu při obsazení území. A v poslední fázi potřebujeme umět přesouvat armádu. Každý problém bude k rozhodnutí vyžadovat odlišnou množinu informací. Například v případě rozestavění armád nás bude zajímat pozice našich armád, nepřátelských armád a bonus za kontinent, ale v případě doplnění armád nás spíše bude zajímat stav armád na hranicích. To nás dostává k myšlence vytvořit více neuronových sítí, kdy každá bude řešit svůj specifický problém. Kromě vstupů, výstupů a toho, co bude neuronová síť řešit, je potřeba si promyslet její topologii. Zkusíme vytvořit dvě různé topologie. První jednodušší a druhou složitější a uvidíme, která z nich bude dosahovat lepších výsledků. Budeme chtít, abychom mohli neuronové sítě využít na libovolné mapě. Pojďme navrhnout 5 neuronových sítí.

5.3.1 SetUp neuronová síť

Neuronovou síť budeme využívat při rozmístění armády. Bude nám říkat, jak moc je dobré dát armádu na dané území. V její kompetenci nebude rozhodování o validitě tahu. Bude pouze rozhodovat o kvalitě tahu. Proto nám stačí jeden výstup s otevřeným intervalem (0,1). Hodnota blíže nule bude označovat špatné území a hodnota blíže jedné bude označovat dobré území. K tomu se nám bude hodit v neuronech aktivační funkce Sigmoida. Nyní potřebujeme nadefinovat množinu vstupů. První musíme vědět, zda je území neutrální anebo už námi obsazené. Takto rozlišíme případy, kdy musíme obsazovat neutrální území a kdy musíme doplňovat armádu do našich území. Při rozhodování nás bude zajímat okolí daného území. Na základě toho můžeme určit, jak důležité je posílit území. Když bychom se rozhodovali jako lidé, tak bychom se nejspíš dívali na stavy armád, počet nepřátelských území a počet našich území. Čím by dané území bylo dál od našeho území, tím bychom tomu dávali menší váhu, protože vzdálená nepřátelská území nás nemohou tolik ohrozit a naše vzdálená území nám nemohou moc pomoci. Na základě toho rozdělíme okolí území do vrstev dle vzdálenosti od daného území. Nultá vrstva bude značit naše území, u kterého se rozhodujeme. První vrstva budou území, která sousedí s naším územím. Druhá vrstva budou území, která jsou ve vzdálenosti dva od našeho území. Vzdálenost budeme definovat jako

velikost nejkratší cesty mezi dvěma vrcholy v grafu. Takto můžeme vytvořit okolí s libovolným počtem vrstev, ale my se omezíme na okolí s počtem vrstev dva. Další vstupy bude tvořit: počet našich území v první vrstvě, počet našich území v druhé vrstvě, počet našich armád v první vrstvě, počet armád v druhé vrstvě, počet nepřátelských území v první vrstvě, počet nepřátelských území v druhé vrstvě, počet nepřátelských armád v první vrstvě a počet nepřátelských armád v druhé vrstvě. Další důležitou informací jsou vlastnosti kontinentu, kde území leží. Určitě by nás zajímalo, jaký bonus obsazený kontinent přináší, jak se bude dobře bránit a kolik území z kontinentu jsme už obsadili. První informací bude celkový počet území na kontinentu. S tím souvisí informace počtu našich území a počtu nepřátelských území na kontinentu. Dále využijeme pár vzorců a hodnot z kapitoly 2. Jedna z nich bude počet území, která musíme obsadit, abychom získali jednu armádu. Druhou informací je počet armád na jedno území na hranici kontinentu a poslední představuje obranný koeficient. Celkem má neuronová síť 16 vstupů a jeden výstup.

Posledním problémem je topologie neuronové sítě. Potřebujeme vytvořit jednodušší a komplexnější topologii. U jednodušší topologie zkusíme použit jednu skrytou vrstvu a jednu výstupní vrstvu. Skrytá vrstva bude obsahovat stejný počet neuronů, jako je vstupů. Výstupní vrstva bude mít pouze jeden neuron. Celkově se neuronová síť bude skládat ze 17 neuronů a 272 synapsí. Oproti tomu komplexnější topologie se bude skládat ze 3 skrytých vrstev a jedné výstupní vrstvy. Výstupní vrstva bude opět obsahovat jeden neuron. První skrytá vrstva bude obsahovat 16 neuronů. Můžeme si představit, že se bude koukat na vstup jako na celek. Druhá skrytá vrstva bude obsahovat 9 neuronů. Tady si představíme, jakoby se dívala pouze na okolí daného území. Poslední třetí skrytá vrstva bude mít 7 neuronů a ta nám bude dávat náhled na vlastnosti kontinentu. Celkem máme 33 neuronů a 470 synapsí.

5.3.2 Exchange Card neuronová síť

Neuronová síť pro výměnu karet bude malá. Nepotřebujeme tolik vstupních informací a rozhodnutí je jednodušší. Výstupem neuronové sítě bude hodnota v otevřeném intervalu (0,1), kdy hodnota menší rovna jak 0,5 bude znamenat nevyměňovat a hodnota vyšší jak 0,5 bude znamenat vyměnit. Budeme mít celkem 4 vstupní informace. První bude dobré znát počet volných armád a kolikátá kombinace karet byla vyměněna. Nakonec bude užitečné mít stav hranic. Kolik našich armád se nachází na hranicích a kolik nepřátelských armád je na hranici. Dle výstupu a vstupních dat opět použijeme aktivační funkci Sigmoidu.

Topologie neuronové sítě bude v obou případech jednoduchá. Na tak málo vstupních informací nemáme moc možností pro vytvoření topologie. Nejjednodušší topologie bude mít jednu skrytou vrstvu a jednu výstupní vrstvu. Skrytá vrstva bude obsahovat 4 neurony a výstupní jeden. Celkem máme 5 neuronů a 20 synapsí. Komplexnější topologie bude taky jednoduchá. Místo jedné skryté vrstvy budeme mít 3. První bude obsahovat 4 neurony a zbylé dvě budou obsahovat po dvou neuronech. Opět máme analogii náhledu na vstup jako celek a pak na jednotlivé části vstupu.

5.3.3 Draft neuronová síť

Doplňování armády bude potřebovat složitější rozhodnutí. Musíme vědět, jak je dobré doplnit armádu na dané území a navíc potřebujeme znát množství armád, které máme doplnit. Na základě toho budeme potřebovat, aby neuronová síť měla dva výstupy. Oba dva výstupy nám budou dávat hodnoty v otevřeném intervalu (0,1). První výstup bude stejný jako u dvou předchozích neuronových sítí. Čím je hodnota blíže 1, tím je dané území lepší. Druhý výstup nám bude říkat, kolik armád máme doplnit. Nechť druhý výstup má hodnotu y, pak výsledná armáda, kterou máme doplnit je rovna $y(volnáArmáda)$. Protože počet armád je celé číslo, musíme výsledek zaokrouhlit na celé číslo. Vždy však minimální hodnota je 1. Na základě výstupu využijeme v neuronech aktivační funkci Sigmoidu. Teď už nám zbývá vymyslet množinu vstupních informací a topologii. První informací, kterou budeme potřebovat, je počet jednotek na daném území. Není potřeba doplňovat armádu na místo, kde je dostatek armády. Zda je na daném území dostatek armád, určíme dle jeho okolí. Okolí budeme popisovat stejně jako u první neuronové sítě a opět vezmeme okolí velikosti 2. Okolí nám dá dalších 8 vstupů a celkem budeme mít 9 vstupů. Další informace nebudeme potřebovat k tomu, abychom se rozhodli, která území posílit.

V topologii neuronové sítě zkusíme použít jednu skrytou vrstvu a jednu výstupní vrstvu. Skrytá vrstva se bude skládat z 9 neuronů a výstupní bude mít 2 neurony. Když propojíme jednotlivé vrstvy neuronů, dostaneme 99 synapsí. U komplexnější topologie zkusíme přidat ještě jednu skrytou vrstvu se stejným počtem neuronů jako má první skrytá vrstva. Ve výsledku budeme mít o 9 neuronů více a celkem 180 synapsí.

5.3.4 Attack neuronová síť

Ve fázi útoku máme dva různé tahy. Jeden je normální útok a druhý je obsazení. Tahy jsou si velice podobné, proto zvolíme pouze jednu neuronovou síť na oba tahy. Pokud budeme útočit, potřebujeme vědět, jestli je dobré provést útok z jednoho území na druhé. K tomu navíc potřebujeme druhou informaci o velikosti útoku. V případě obsazení nepotřebujeme znát, jak moc je dobré dané území obsadit, protože už jsme jej obsadili, ale musíme znát počet armád k přesunu. Žádné území nesmí být bez armády. Můžeme využít už jeden existující výstup neuronové sítě. Druhý výstup nám bude říkat s jakou armádou útočit a kolik armády přesunout. Na základě toho budeme mít dva výstupy. Opět se hodnota výstupů bude pohybovat v otevřeném intervalu (0,1) a využijeme aktivační funkci Sigmoidu. Význam prvního výstupu bude spočívat v informaci, zda útočit z území A na území B. Pokud hodnota bude vyšší jak 0,5, provedeme útok, jinak nebudeme útočit. V případě obsazení bude ignorován. Druhý výstup bude značit velikost útoku, kdy možnosti jsou 1,2 a 3. Velikost 1 bude odpovídat hodnotě výstupu menší jak 0,33. Velikost 2 bude odpovídat hodnotě výstupu v intervalu [0,33,0,66) a poslední velikost 3 bude odpovídat hodnotě větší rovno 0,66. Při obsazení bude mít druhý výstup jiný význam. Bude určovat velikost armády k přesunu. Nechť y je náš výstup neuronové sítě, pak velikost armády k přesunu spočítáme následovně:

$$armadaKPresunu = (armadaNaUzemi - 1 - velikostUtoku)y + velikostUtoku$$

Hodnota *velikostUtoku* je velikost útoku, která dobyla území. Výsledek opět zaokrouhlíme a tím získáme výslednou armádu. Otázka je, jaké vstupy neuronové síti poskytnout. Určitě bychom chtěli první dobýt kontinent, který je nejvýhodnější. Na druhou stranu nechceme útočit tam, kde je silný odpor. Na základě toho poskytneme neuronové síti informace o počtu armád na útočícím území, počtu armád na bránícím území, okolí útočícího území a vlastnosti kontinentu bránícího území. Okolí vezmeme velikosti 2 a vlastnosti kontinentu vezmeme naše známé, které jsme využili v první neuronové síti. Celkově budeme mít 17 vstupů. Jestli se jedná o útok nebo obsazení, bude moci neuronová síť rozlišit pomocí počtu armád na bránícím území. Bránící území bez armády bude značit obsazení.

Teď se můžeme podívat na topologii. Využijeme náš klasický přístup k topologii neuronové sítě. Nejprve vytvoříme jednoduchou topologii. Budeme potřebovat jednu skrytou vrstvu a jednu výstupní vrstvu. Ve skryté vrstvě budeme mít 17 neuronů. Počet odpovídající množství vstupů. Výstupní vrstva bude obsahovat 2 neurony. Pro každý výstup jeden neuron. Složení takové topologie je 19 neuronů a 323 synapsí. Komplexnější topologie bude mít více skrytých vrstev. První bude mít opět 17 neuronů a v následujících vrstvách využijeme analogii, že každá se zaobírá jedním problémem lokálně. Tedy druhá vrstva bude mít 10 neuronů. Kouká se na dané dvě území a okolí. Třetí vrstva bude obsahovat 7 neuronů sledujících vlastnosti kontinentu. Výstupní vrstva zůstává stejná. Nakonec dostáváme 36 neuronů a 543 synapsí.

5.3.5 Fortify neuronová síť

Poslední neuronovou sítí, kterou potřebujeme, je neuronová síť k přesunu jednotek. Budeme potřebovat dva výstupy. Oba dva výstupy budou dávat hodnoty v otevřeném intervalu (0,1). Na základě toho využijeme aktivační funkci Sigmoidu. První výstup bude značit, jak dobré je přesunout armádu z území A do území B. Druhý výstup nám pak řekne, kolik armády přesunout. Nechť y je hodnota druhého výstupu, pak armádu k přesunu získáme zaokrouhlením součinu $y(armadaNaUzemiA - 1)$. Nejméně vždy přesuneme jednu armádu. Přesouvat budeme z území A do území B, které má nejvyšší první hodnotu výstupu a to právě tehdy a jen tehdy, když hodnota je vyšší než 0,5. Zbývá nadefinovat množinu vstupů. V tomto případě budeme potřebovat nejvíce vstupů. Musíme znát počet armád na území A a na území B. Dále se musíme podívat na okolí obou území. V obou případech se budeme dívat na okolí velikosti 2. Na základě těchto informací bychom měli být schopni dobře rozhodnout o přesunu armády. Výsledný počet vstupů je 18.

V případě topologie bude největší neuronovou sítí z našich sítí. Nejprve navrhneme jednodušší topologii. Budeme mít pouze jednu skrytou vrstvu a jednu výstupní vrstvu. Ve skryté vrstvě budeme mít 18 neuronů a ve výstupní vrstvě 2 neurony. Po sečtení dostáváme 20 neuronů a 360 synapsí. Komplexnější topologie bude daleko větší. V první vrstvě necháme 18 neuronů, ale přidáme další 2 skryté vrstvy. V obou vrstvách bude 9 neuronů. Pro každé území a jeho okolí jedna vrstva. Výstupní vrstva bude stejná. Ve výsledku dostáváme 38 neuronů a 585 synapsí.

5.3.6 Implementace Agenta

Když už jsme navrhli všechny neuronové sítě, potřebujeme je naimplementovat. Protože píšeme všechno v programovacím jazyce C#, zkusíme v něm napsat i neuronové sítě. K tomu nám poslouží machine learningový framework Accord.NET[1]. Konkrétně z něj využijeme knihovny Accord.Neuro, Accord.Genetic a jejich závislosti Accord.Statistics a Accord.Math. Knihovna Accord.Neuro obsahuje implementaci neuronové sítě tak, jak ji známe z kapitoly 4. Neuronovou síť implementuje třída ActivationNetwork, která obsahuje ActivationLayer obsahující ActivationNeuron. V konstruktoru nám stačí zadat aktivační funkci, počet vstupů a pak můžeme pro každou vrstvu nadefinovat počet neuronů. Na základě toho vytvoříme všechny naše neuronové sítě. Třída ActivationNetwork nám také poskytuje uložení a načtení neuronové sítě z/do souboru. To se nám bude hodit při využívání naučených neuronových sítí.

Agent bude obsahovat všech 5 neuronových sítí. V každé fázi připraví vstupy pro neuronovou síť a bude provádět tahy na základě jejího rozhodnutí. V úvodu hry vždy najde všechny možnosti, kam rozestavit armádu a pro každou možnost připraví vstup. Všechny vstupy postupně předá neuronové síti a vybere možnost s nejvyšší hodnotou, pro kterou vytvoří tah. Ve fázi doplňování armády bude postupovat podobně, ale nejprve zkontroluje kombinaci karet. Jestliže je potřeba, vymění potřebné množství karet. Pokud má nějakou další kombinaci karet, připraví a předá vstup neuronové síti určené pro výměnu karet. Na základě jejího rozhodnutí, pak vymění nebo nevymění karty. Po výměně karet začne doplňovat armádu. Bude opakovat proces, dokud nevypotřebuje všechny volné armády. Ve fázi útoku bude proces odlišný. Najde všechny možnosti odkud útočit a pro každou možnost najde všechny možnosti kam útočit. Pro každou dvojici útočník-obránce vytvoří vstup a předá ho neuronové síti. Pokud rozhodnutí bude útočit, zaútočí s armádou, kterou určila neuronová síť. Po útoku se opět zeptá na stejnou dvojici. Bude útočit tak dlouho, dokud nepřijde rozhodnutí o neútočení. Tento cyklus opakuje pro každou validní dvojici útočník-obránce. Mezi útočením může dojít k obsazení. Při obsazení sestaví stejný vstup a zeptá se stejné neuronové sítě, kolik má přesunout armády. V poslední fázi přesunutí armády najde opět všechny možnosti přesunu armády z/do území a pro každou možnost vytvoří vstup. Vstupy postupně předá příslušné neuronové síti a vybere tu nejlepší možnost. Pokud hodnota nejlepší možnosti nepřesahuje 0,5, nebude přesouvat žádnou armádu. Tímto způsobem naimplementujeme celý potřebný interface IPlayer.

5.3.7 Učení neuronových sítí

V kapitole 4 jsme si povídali o evolučních algoritmech jako o způsobu učení neuronových sítí. Tyto algoritmy využijeme, abychom naučili našich 5 neuronových sítí. K tomu nám poslouží knihovna Accord.Genetic. Knihovna obsahuje třídu Population, která pak provádí evoluci. V konstruktoru třídy musíme zadat velikost populace, chromozom, fitness funkci a selekci. Velikost populace zvolíme 100. Abychom mohli předat chromozom, potřebujeme třídu implementující IChromosome. V knihovně Accord.Genetic je připraveno několik základních chromozomů. My využijeme chromozom DoubleArrayChromosome, který představuje

[1] http://accord-framework.net/

pole reálných čísel. Pole reálných čísel bude představovat váhy našich neuronových sítí. Konstruktor chromozomu vyžaduje generátory náhodných reálných čísel a velikost pole. Jako generátory náhodných reálných čísel využijeme třídu z knihovny Accord.Math ZigguratUniformOneGenerator. Velikost pole se bude rovnat součtu počtu vah a počtu thresholdů všech neuronových sítí. Další konstruktor místo velikosti pole bere rovnou vytvořené pole reálných čísel. Tento konsturktor se může hodit v učení už nějak naučených neuronových sítí. Dále potřebujeme vytvořit fitness funkci. K tomu musíme vytvořit vlastní implementaci interfacu IFitnessFunction, která vyžaduje pouze implementaci metody Evaluate. Musíme nějakým způsobem ohodnotit, jak jsou dané neuronové sítě dobrými jedinci. Na základě toho, že nemáme žádnou databázi správných tahů nebo nevíme, které tahy jsou dobré a které špatné, nemůžeme jen tak spočítat fitness funkci. Musíme nechat neuronové sítě, ať se učí hraním proti jiným agentům. Výsledek fitness funkce pak bude $\frac{pocetVytezstvi}{pocetHer}$. Zbývá zvolit metodu selekce. V knihovně máme naimplementovanou selekci elity, kterou představuje třída EliteSelection. Nyní můžeme vytvořit objekt Population. Než spustíme evoluci, můžeme nastavit další parametry jako je pravděpodobnost křížení a mutace. Pro spuštění evoluce jedné generace stačí zavolat metodu RunEpoch.

5.4 MCTS agent s využitím neuronové sítě

U prvního MCTS agenta jsme museli vymýšlet heuristiku, abychom ho mohli reálně použít. Když máme vytvořené neuronové sítě, můžeme je zkusit použít místo heuristiky. Dostatečně naučené neuronové sítě nám mohou odstranit mnoho špatných tahů. MCTS spolu s neuronovými sítěmi by mohlo dosahovat daleko lepších výsledků, protože ručně psaná heuristika odpovídá naší znalosti hry. Neuronové sítě využijeme při generování stavů. Každá neuronová síť má jeden výstup, který určuje, jak je dobré táhnout na daném území. Tuto hodnotu využijeme při filtraci tahů. Definujeme si mez, která rozdělí tahy na dobré a špatné. V úvodu hry při rozmístění armád otestujeme všechny možnosti a MCTS předáme jen ty, které přesáhnou naši mez. V první fázi kola při doplňování armády opět vybereme všechny možnosti a předáme je příslušné neuronové síti. Ta nám provede filtraci tahů. Rozdíl oproti původnímu MCTS je, že nebudeme provádět částečné uspořádání a necháme všechnu práci na neuronové síti. Pro každý možný tah vybereme další možné tahy a tak vygenerujeme všechny stavy. V druhé fázi při útoku budeme postupovat stejně jako v původním MCTS. Rozdíl bude v tom, že o útočení a o tom, jak velký útok provést, bude rozhodovat neuronová síť. Útočit budeme jen, pokud hodnota prvního výstupu neuronové sítě přesáhne naši mez. V případě, že rozhodne útočit, doplníme ještě možnost neútočit. Jinak útočíme na jedno území dokud neuronová síť neřekne stop. Princip generování stavů zůstane stejný včetně částečného uspořádání. Dojde-li k obsazení území, neuronová síť zvolí, kolik přesně armády přesunout. V poslední fázi při přesunutí armády budeme postupovat obdobně jako v první fázi. Najdeme všechny možnosti přesunu a zeptáme se neuronové sítě, jestli je dobré přesunout armádu a kolik armád přesunout. Všechny možnosti, které překročí mez zpracujeme. Nakonec přidáme ještě možnost nepřesunout žádnou armádu. S pomocí principu v původním MCTS a za použití neuronových sítí vygenerujeme všechny možné stavy. Další průběh algoritmu nebudeme měnit a zůstane stejný jako v původním MCTS.

6. Experimenty

6.1 Evoluce neuronových sítí

Abychom naučili naše neuronové sítě hrát, musíme provést evoluci. V kapitole 5.3.7 jsme se podívali, jak implementovat evoluci a jaké parametry můžeme nastavit. Než spustíme naši evoluci musíme vyřešit všechny parametry. Velikost populace zvolíme 100. Pravděpodobnost křížení bude 0,5 a pravděpodobnost mutace nastavíme na 0,01. Velikost pole reálných čísel, které reprezentuje chromozom, se bude rovnat součtu vah synapsí a thresholdů všech neuronových sítí. V případě jednoduché topologie má pole velikost 1146 a v případě komplexnější topologie má velikost 1944. Nakonec nám zbývá určit parametry fitness funkce a zvolit počet generací. U fitness funkce musíme rozhodnout, proti komu bude učená neuronová síť hrát, kolik her odehraje a na jaké mapě. Na základě odehraných 500 her pro 3 hráče agenta s nenaučenou neuronovou sítí proti náhodným agentům (viz Tabulka 6.1 a 6.2) zvolíme jako nepřítele agenta s nenaučenou neuronovou síť. Od náhodného agenta by se neuronová síť mnoho nenaučila. Nejprve naučíme jednu neuronovou síť a pak budeme hrát proti naučené neuronové síti. Díky tomu budeme mít čím dál více silnějšího nepřítele a budeme moci toho více neuronové sítě naučit. Hry budou probíhat na náhodně generované mapě s 21 územími. Potřebujeme, aby hry netrvaly příliš dlouho, abychom jich mohli odehrát co nejvíce. Hra na klasické mapě se 42 územími trvá 1 sekundu a díky tomu by učení trvalo několik dní. Čím více her odehrajeme ve fitness funkci, tím přesněji budeme moc ohodnotit daného jedince. Problémem je čas. Ohodnocení fitness funkcí bude trvat tak dlouho, než se odehrají všechny hry. Kdybychom zvolili počet her 500, tak by jedno ohodnocení trvalo okolo 15 sekund a to za předpokladu, že omezíme čas hry na 1 sekundu. V tomto případě by evoluce 100 generací populace o velikosti 100 trvala přibližně 2 dny. Evolucí musíme provést víc, proto musíme snížit počet her na 200. Všechny parametry máme nastavené a můžeme spustit evoluce.

	Agent s neuronovou sítí	Náhodný agent 1	Náhodný agent 2
Vítězství	424	38	38

Tabulka 6.1: Neuronová síť s jednoduchou topologií

	Agent s neuronovou sítí	Náhodný agent 1	Náhodný agent 2
Vítězství	471	13	16

Tabulka 6.2: Neuronová síť s komplexnější topologií

V první evoluci nastavíme počet generací na 50. Máme nenaučené neuronové sítě, takže předpoklad je, že učení půjde rychle. V druhé evoluci bude učení náročnější, protože budeme mít už naučené neuronové sítě. Na základě toho nastavíme počet generací na 60. Když se podíváme na výsledky evolucí v příloze A.2 zjistíme, že potřebujeme nastavit vyšší počet generací. Zkusme ve třetí evoluci

nastavit počet generací na 80 a protihráče nastavíme druhé naučené neuronové sítě. Vidíme, že stále máme málo generací. Zkusme provést ještě čtvrtou evoluci, která bude mít 100 generací a protihráče třetí naučené neuronové sítě.

V příloze A.2 najdeme dva druhy výsledků evolucí. Jedny výsledky patří neuronovým sítím s jednoduchou topologií a druhé patří s komplexnější topologií. Byly provedeny 2 pokusy evolucí. Můžeme vidět, že neuronové sítě s komplexnější topologií dosahují lepších výsledků a lépe se učí. Graf čtvrté evoluce neuronových sítí s jednoduchou topologií v prvním pokusu dosáhl maxima okolo 0,5 a dál neroste. Dokonce graf třetí evoluce v prvním pokusu je na tom podobně. V druhém pokusu neuronové sítě s jednoduchou topologií dopadly trochu lépe než v prvním pokusu, ale také dopadly lépe neuronové sítě s komplexnější topologií v první evoluci. Díky tomu mohou být další výsledky evolucí komplexnější topologie horší. Na základě toho, že neuronové sítě s jednoduchou topologií dosahují horších výsledků než neuronové sítě s komplexnější topologií, to může znamenat, že neuronové sítě s jednoduchou topologií nejsou ani schopny se naučit dobře hrát. Hra Risk může být pro ně příliš složitá a může vyžadovat daleko komplexnější topologii. Kromě porovnání výsledků evolucí najdeme v příloze B výsledky odehrání 500 her mezi neuronovými sítěmi z různých evolucí. Je zajímavé, že neuronové sítě s jednodušší topologií z prvního pokusu poráží neuronové sítě s komplexnější topologií. Na základě výsledků evoluce bychom spíše předpokládali, že neuronové sítě s komplexnější topologií budou lepší. V druhém pokusu jsou zase lepší neuronové sítě s komplexnější topologií. Když porovnáme neuronové sítě z dvou pokusů, tak nejlépe na tom jsou neuronové sítě s komplexnější topologií ze čtvrté evoluce. Porážejí všechny neuronové sítě, ale mají trošku problém s neuronovými sítěmi s komplexnější topologií z prvního pokusu ze čtvrté evoluce (viz Tabulka 6.3 a 6.4). V rámci experimentu by chtělo sérii evolucí a následné porovnání výsledků soubojů zopakovat vícekrát. Na základě času potřebného pro jeden pokus nebylo možné pokusy vícekrát zopakovat, a proto máme pouze dva pokusy.

	Agent s CNN 31	Agent s CNN 32	Agent s CNN 32
Vítězství	226	127	147

Tabulka 6.3: Výsledky souboje komplexnější topologie

	Agent s CNN 32	Agent s CNN 31	Agent s CNN 31
Vítězství	160	172	168

Tabulka 6.4: Výsledky souboje komplexnější topologie

6.2 Paměťové nároky

Vytvořili jsme 3 druhy různých agentů. Pokud započítáme i náhodného agenta, máme celkem 4 agenty. Někteří paměti moc nepotřebují, ale někteří ji vyžadují velké množství. Pojďme se podívat, kolik jednotliví agenti potřebují paměti (viz Tabulka 6.5). Náhodný agent nepotřebuje skoro žádnou paměť. Jediné, kdy spotřebuje nějakou malou část paměti, je při vytvoření seznamu možností. Takovou malou spotřebu paměti lze zanedbat. Obdobně je na tom agent s neuronovou

sítí. V paměti si udržuje pouze neuronové sítě a opět vytváří seznam možností, z kterých připravuje vstupy pro neuronové sítě. Spotřeba paměti je o něco větší, ale stále zanedbatelná. Problém nastává u MCTS agenta. MCTS agent si buduje strom, který si musí uchovávat v paměti. Pokud existuje mnoho možných následujících stavů, spotřeba paměti extrémně vzroste. Díky tomu nelze MCTS agenta použít na větší mapy, které obsahují více než 18 území. Paměť není jediná, která tomu brání. V případě mapy obsahující 18 území budeme potřebovat nejméně 8 GB RAM. Spotřebu jednotlivých tří agentů můžeme vidět níže. Zkratkou NN je myšleno neuronová síť. MCTS agent s neuronovými sítěmi využil více než 8 GB RAM, proto nemá uvedeno konečné číslo. Problémem jsou málo naučené neuronové sítě, které dávají mnoho dobrých následujících tahů. MCTS agent s neuronovými sítěmi dokáže zvládnout maximálně mapu obsahující 9 území a to jedině, když využívá neuronové sítě s komplexnější topologií ze čtvrté evoluce. Zajímavé je, že ve hrách se zdály být lepší neuronové sítě s jednodušší topologií. Jediný agent, který by mohl být využit na libovolné mapě, je agent s neuronovými sítěmi.

	MCTS agent	MCTS agent s NN	Agent s NN
Pamět (max)	6 GB	∞	0 GB

Tabulka 6.5: Spotřeba paměti

6.3 Reakční doba

Reakční doba je další element, který rozhoduje o použitelnosti agenta (viz Tabulka 6.6). Pro představu otestujeme i náhodného agenta, abychom věděli reakci agenta, který neprovádí žádné výpočty. Průměrné časy náhodného agenta v různých fázích hry se pohybují vždy v rámci 10^{-6} sekund. S průměrnými časy se nejvíce náhodnému agentovi přiblížil agent s neuronovými sítěmi. Výpočet neuronových sítí je lehce znatelný oproti náhodnému agentovi, ale lidský hráč by nerozpoznal rozdíl. MCTS agenti mají první dva časy vždy větší, protože provádí výpočet a pak jen odehrají tahy. Nejhorší čas má MCTS agent s neuronovou sítí, kdy se propojil výpočet MCTS a neuronové sítě. Musíme poznamenat, že MCTS provádí 1000 iterací. Kdybychom chtěli větší mapu, museli bychom zvýšit počet iterací. To by znamenalo zvýšení času reakce a lidský hráč by už mohl rozpoznat pomalou odezvu agenta. Nejlépe je na tom opět agent s neuronovými sítěmi.

časy (s)	MCTS agent	MCST agent s NN	Agent s NN
Rozmístění	0,74	0,98	0,00021
Doplnění	0,97	3,30	0,00029
Útok	0,0001	0,00016	0,00053
Přemístění	0,00003	0,0001	0,00039

Tabulka 6.6: Reakční doby agentů

6.4 Souboj agentů

V poslední řadě porovnáme inteligenci agentů. Necháme hrát všechny 3 typy agentů proti sobě. Abychom mohli porovnat i MCTS agenta s neuronovými sítěmi, necháme je hrát na náhodně vygenerované mapě obsahující 9 území. Celkem odehrají 100 her. Nejprve je však necháme hrát ještě proti náhodnému agentovi, abychom ověřili, že jsou alespoň trochu inteligentní. Všechny výsledky her najdeme v příloze B. Můžeme vidět, že všichni tři agenti poráží náhodného agenta. To znamená, že dokáží hrát relativně inteligentně. Necháme-li hrát agenty mezi sebou, uvidíme, že vítězem se stal MCTS agent (viz Tabulka 6.7). Za ním se umístil MCTS agent s neuronovou sítí a jako poslední skončil agent s neuronovou sítí. Na základě výsledků MCTS agenti dokáží hrát nejlépe.

	MCTS agent	MCTS agent s NN	Agent s NN
Vítězství	45	35	20

Tabulka 6.7: Výsledky souboje

Zahrajeme-li si proti agentům, zjistíme, že hra proti MCTS agentům je velice zdlouhavá. Čekání na odehrátí tahu MCTS agenta je dosti znatelné a hra může po chvíli čekání začít být nudná. Na druhou stranu hraní MCTS agentů připomíná hraní lidí. Lépe reagují na hráčovy tahy a pokaždé hrají jinak, takže se hra může zdát být zajímavější. Oproti tomu agent s neuronovými sítěmi má naučenou jednu strategii a tu aplikuje na každou hru. Někdy agentovi strategie vyjde, ale někdy prohraje. Výhodou tohoto agenta je rychlost. Tahy provádí okamžitě, a proto hráč nemusí čekat a více si užije hraní. Ovšem hrát proti stejné strategii může taky začít nudit. Všichni agenti jsou schopni porazit lidského hráče začátečníka na mapě obsahující 9 území. S rostoucím počtem území kvalita strategie agenta s neuronovými sítěmi klesá a MCTS agenti větší mapy nezvládají.

Závěr

Zhodnocení

V práci jsme nejprve připravili rozhraní hry a posléze ho naimplementovali. K dispozici jsou 2 implementace her. Jedna slouží pro bežné hraní, obsahuje kontroly, aby hráči nemohli podvádět. Druhá implementace slouží k simulaci hry, je naimplementovaná bez všech kontrol a přímo provádí tahy hráčů. Rozhraní je navržené tak, aby si mohl kdokoliv snadno vytvořit svou vlastní implementaci hry, a v ní využít již existující agenty, nebo vlastní implementaci hráče. Nad implementací rozhraní hry byl vytvořen singleplayer, kde si uživatel může zahrát proti libovolnému agentovi. Na základě vlastností agentů někteří mohou hrát pouze na malých mapách. Uživatel si může vybrat mezi klasickou mapou nebo náhodně generovanou. Kromě singleplayeru může hráč vyzkoušet multiplayer, kdy je připraven server a klient aplikace. Díky serveru si může kdokoliv naimplementovat vlastního klienta, který musí pouze dodržovat pravidla síťové komunikace.

Nakonec jsme se pokusili naimplementovat 3 agenty a 1 hloupějšího pomocného agenta. Nejlépe hrajícím agentem je MCTS agent, ale jeho využití je omezené. Dokáže hrát pouze na malých mapách a jeho reakční doba je pomalá. MCTS agent s neuronovými sítěmi dokáže hrát také dobře, ale jeho rychlost a spotřeba paměti je ještě horší než MCTS agenta. Oproti tomu agent s neuronovými sítěmi sice nehraje tak dobře jako MCTS agent, ale jeho reakční doba je nejrychlejší a dokáže hrát na libovolně velké mapě. Je to jediný agent, který může být využit ve hře s klasickou mapou. Při vývoji agenta s neuronovými sítěmi jsme zkusili navrhnout 2 topologie neuronových sítí. U každé topologie jsme provedli 4 evoluce, 2 pokusy a následně je srovnaly. Komplexnější topologie dosahovala daleko lepších výsledků v evoluci, ale nakonec v prvním pokusu byla ve hrách poražena jednodušší topologií. Až v druhém pokusu dokázala komplexnější topologie porazit jednodušší.

Budoucí práce

Díky připravenému rozhraní hry lze snadno experimentovat s implementací různých pravidel hry a agentů ke hře. Další zajímavou prací by mohl být vývoj lepší heuristiky v MCTS. Na základě toho by MCTS mohlo zvládat větší mapy. K tomu by i mohlo dopomoci vhodné zakódování stavu do pár integerů spolu s využitím chytřejšího hráče v simulaci. V případě většího množství času by mohlo být zajímavým srovnáním ručně psaná heuristika s heuristikou využívající naučenější neuronové sítě. V oblasti neuronových sítí je mnoho volného prostoru pro experimentování. Omezíme-li se na mapu s pevným počtem území, mohlo by být zajímavé předávat neuronové síti stav každého území a jejich spojení. Výstupem neuronové sítě by mohl být konkrétní tah. Budeme-li pracovat s proměnlivým počtem území na mapě, lze experimentovat s reprezentací okolí. Okolí můžeme zvětšovat, aby neuronová síť měla větší přehled o dění na mapě. Tím, že je hra poměrně složitá, existuje mnoho možností, které lze vyzkoušet.

Seznam zdrojů

[1] Ultra Board Games. Risk game rules. http://www.ultraboardgames.com/risk/game-rules.php, May 2018. Accessed on 2018-05-01.

[2] Erik Arneson. History of risk board game. https://www.thesprucecrafts.com/history-of-risk-412339, April 2017. Accessed on 2018-05-02.

[3] Jonathan Reem. What is a good strategy for risk (the board game)? https://www.quora.com/What-is-a-good-strategy-for-Risk-the-board-game, May 2013. Accessed on 2018-05-04.

[4] Data Genetics. Risk analysis. http://datagenetics.com/blog/november22011/, November 2011. Accessed on 2018-05-03.

[5] Michael Wolf. *An Intelligent Artificial Player for the Game of Risk*. Darmstadt, 2005. Diploma Thesis. Darmstadt University of Technology. Department of Computer Science.

[6] Guillaume Maurice Jean-Bernard Chaslot. *Monte-Carlo Tree Search*. Maastricht, 2010. Dissertation Thesis. Universiteit Maastricht.

[7] Simon Haynkin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2 edition, 1999.

[8] Mgr. Kristína Medalová. Neuron a jeho stavba. https://www.mentem.cz/blog/neuron/, September 2015. Accessed on 2018-05-03.

[9] Halberdo. Neuron-no labels2.png. https://en.wikipedia.org/wiki/File:Neuron-no_labels2.png, June 2009. Accessed on 2018-05-03.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson, 2009.

[11] Geetika saini. Artificial neural network. https://upload.wikimedia.org/wikipedia/commons/b/b6/Artificial_neural_network.png, March 2017. Accessed on 2018-05-02.

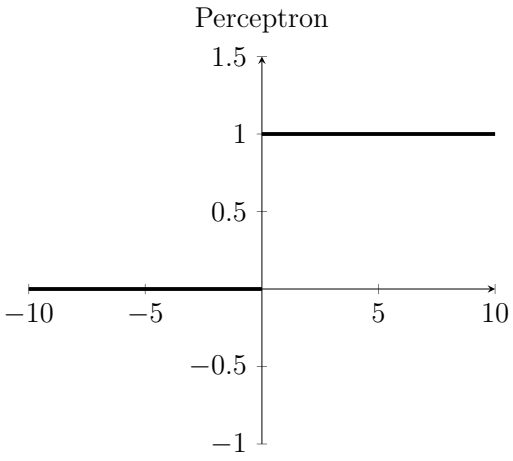
[12] Zbigniew Michalewicz David B. Fogel. *How to Solve It: Modern Heuristics*. Springer Berlin Heidelberg, 2010.

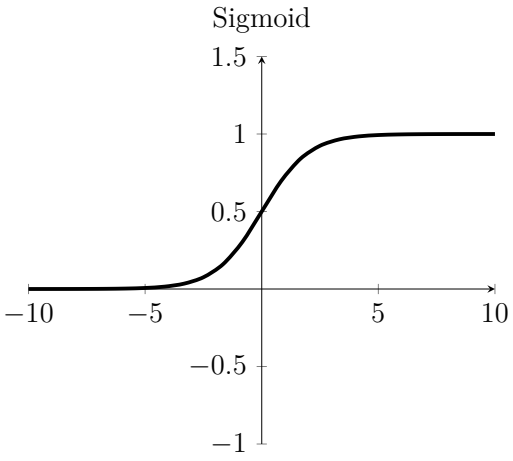
Seznam obrázků

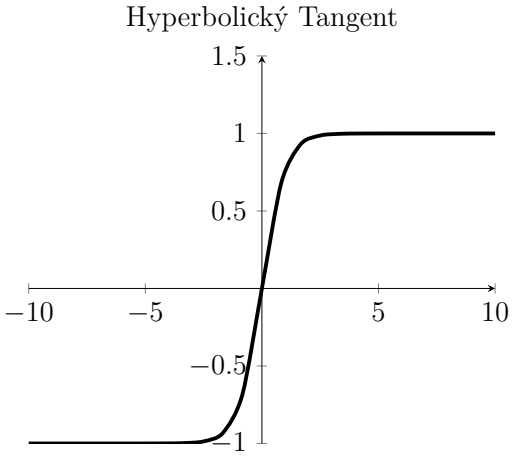
Seznam tabulek

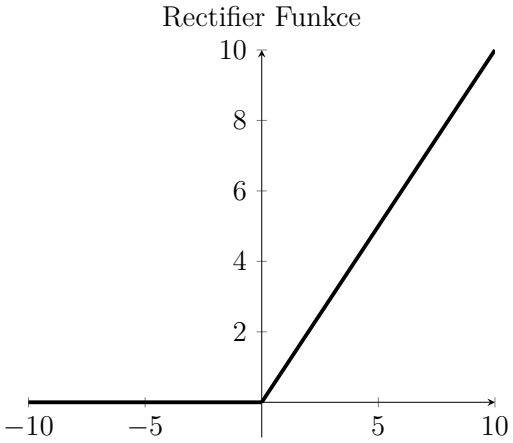
A. Příloha - Grafy

A.1 Aktivační funkce



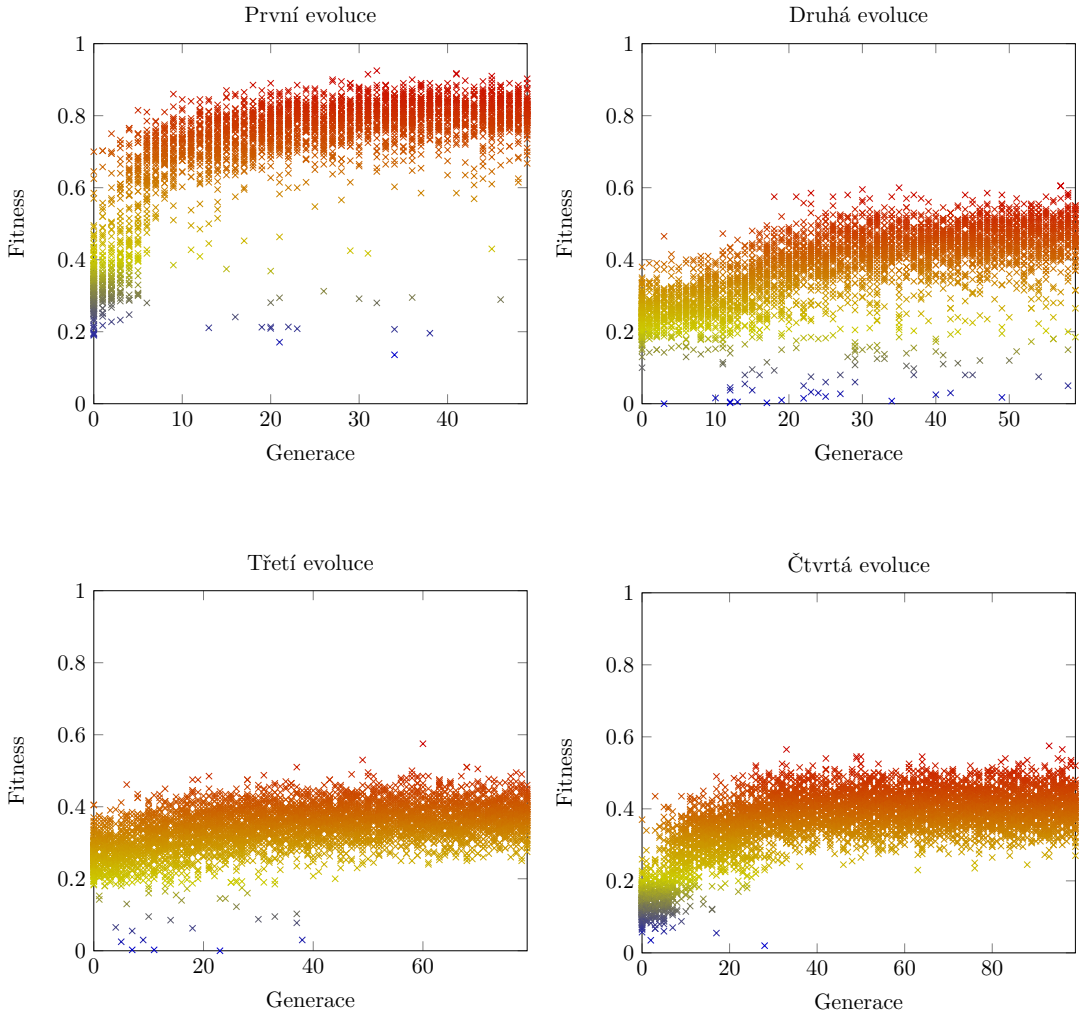




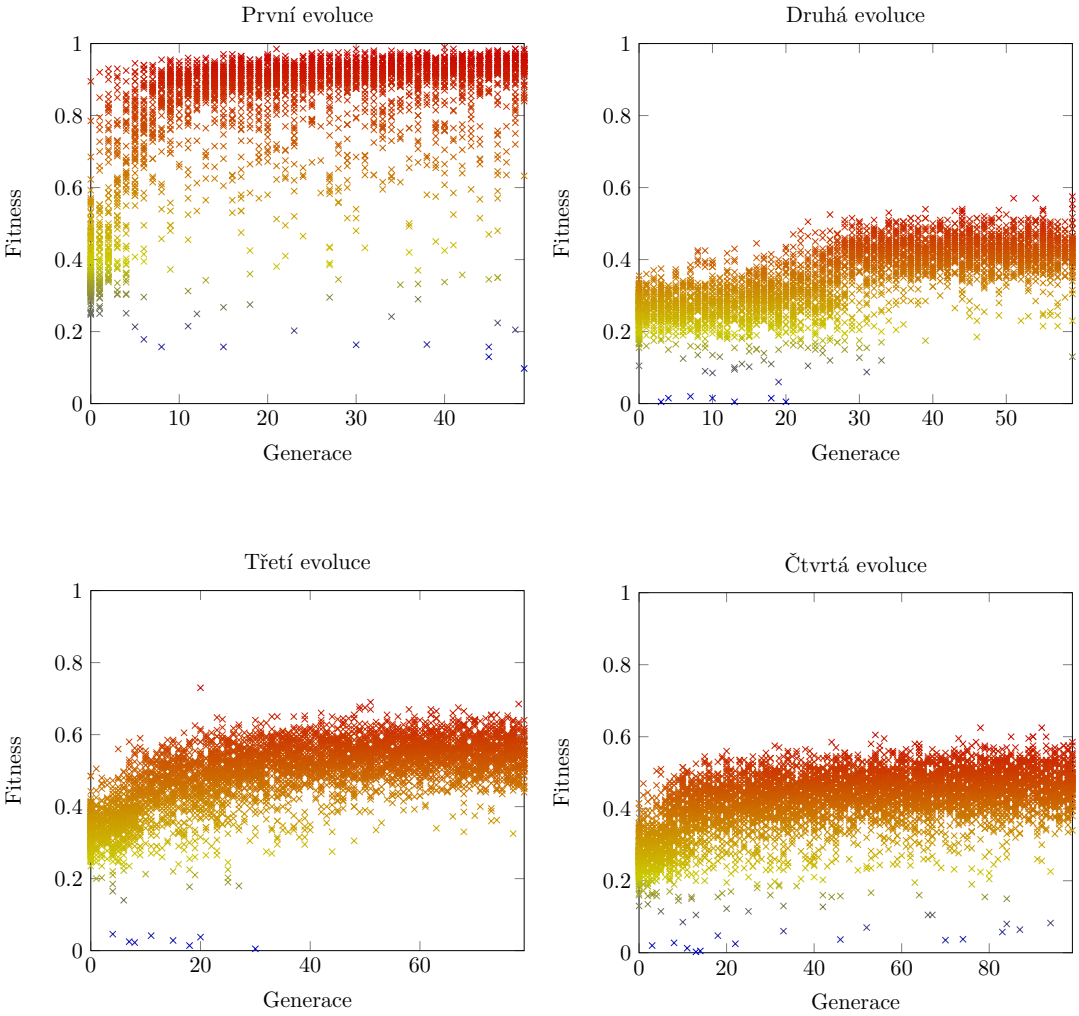


A.2 Výsledky Evoluce

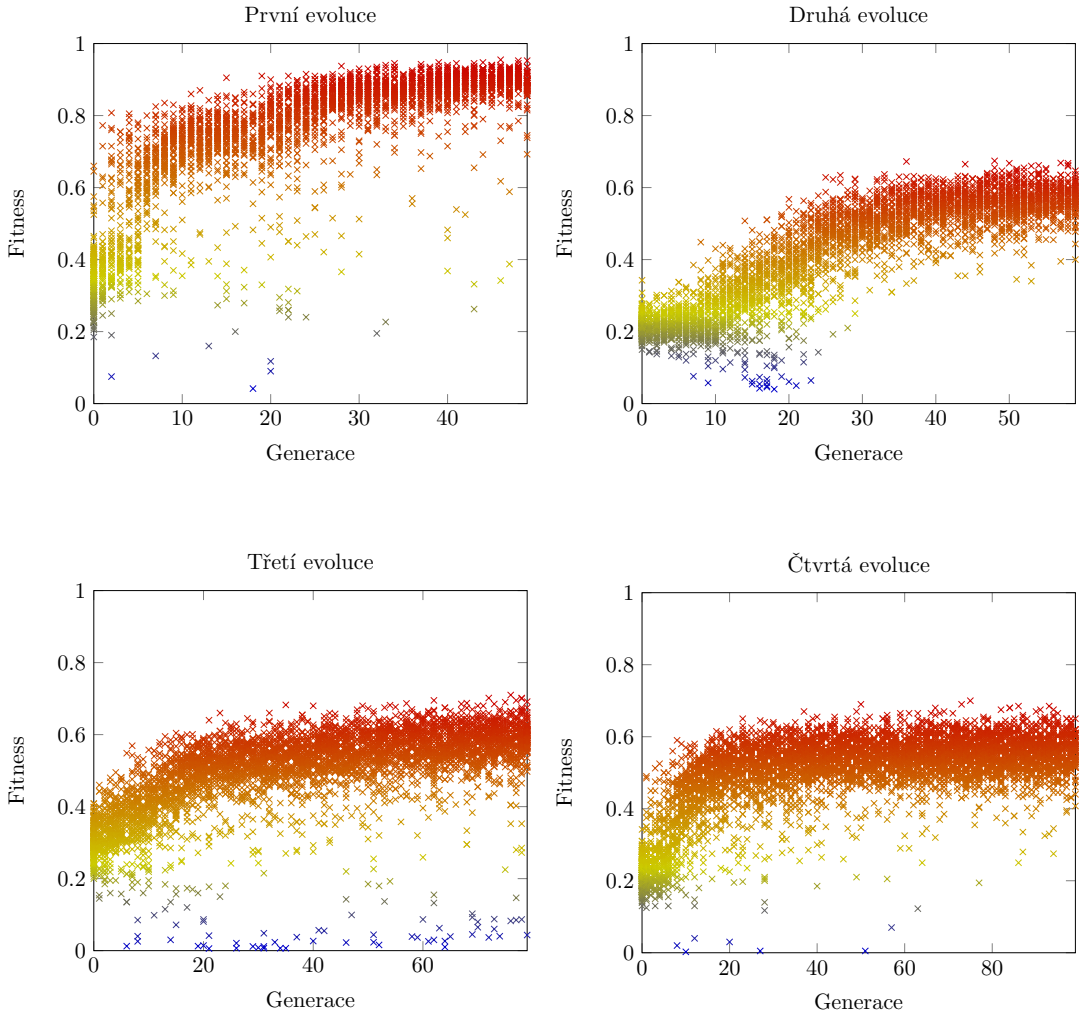
Jednoduchá topologie 1. pokus



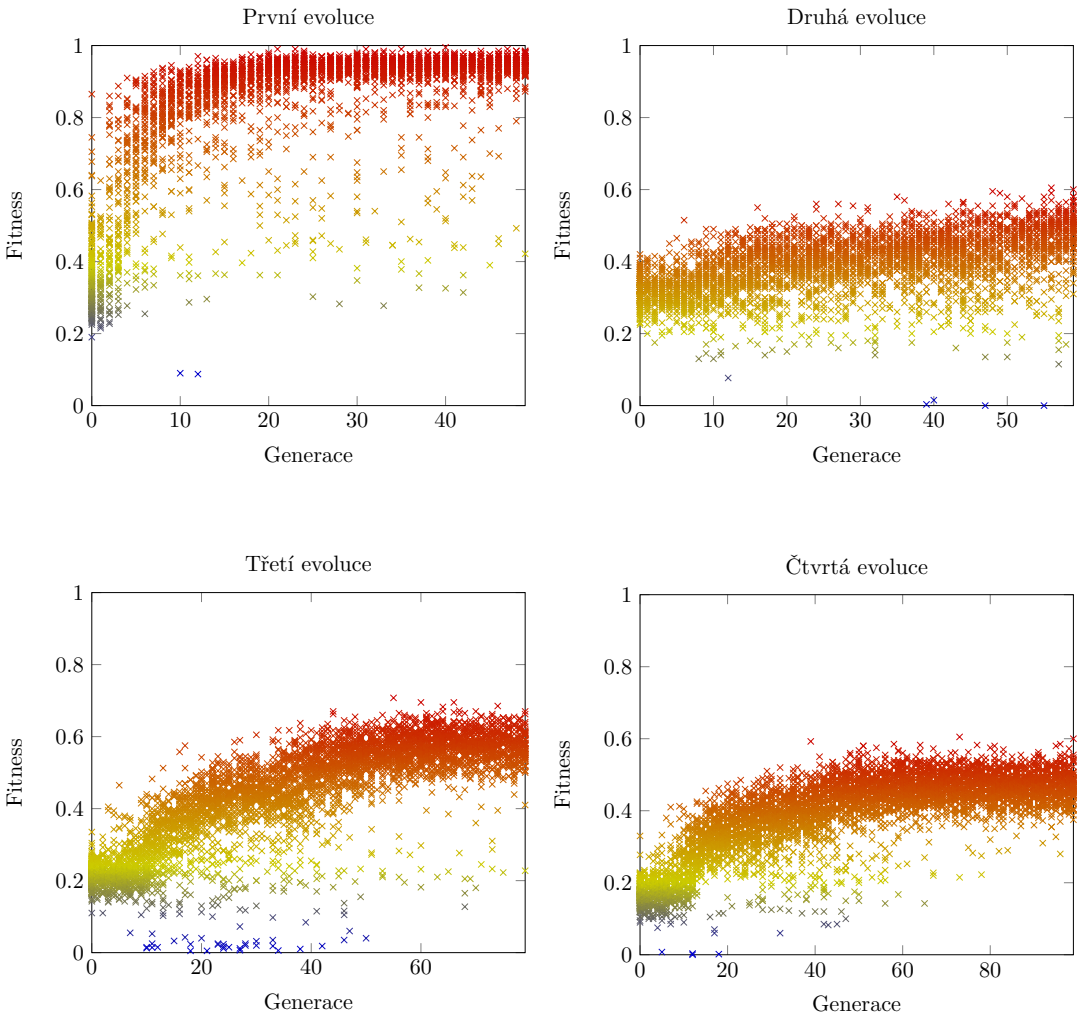
Komplexní topologie 1. pokus



Jednoduchá topologie 2. pokus



Komplexní topologie 2. pokus



B. Příloha - Výsledky her

- NN -1 - nenaučené neuronové sítě
- NN 0X - neuronové sítě po první evoluci v X-tém pokusu
- NN 1X - neuronové sítě po druhé evoluci v X-tém pokusu
- NN 2X - neuronové sítě po třetí evoluci v X-tém pokusu
- NN 3X - neuronové sítě po čtvrté evoluci v X-tém pokusu
- CNN -1 - nenaučené neuronové sítě
- CNN 0X - neuronové sítě po první evoluci v X-tém pokusu
- CNN 1X - neuronové sítě po druhé evoluci v X-tém pokusu
- CNN 2X - neuronové sítě po třetí evoluci v X-tém pokusu
- CNN 3X - neuronové sítě po čtvrté evoluci v X-tém pokusu

Neuronové sítě s jednoduchou topologií

	Agent s NN -1	Náhodný agent 1	Náhodný agent 2
Vítězství	424	38	38

	Agent s NN 01	Náhodný agent 1	Náhodný agent 2
Vítězství	445	28	27

	Agent s NN 01	Agent s NN -1	Agent s NN -1
Vítězství	427	35	38

	Agent s NN 11	Náhodný agent 1	Náhodný agent 2
Vítězství	457	23	20

	Agent s NN 11	Agent s NN -1	Agent s NN -1
Vítězství	331	94	75

	Agent s NN 11	Agent s NN 01	Agent s NN 01
Vítězství	180	166	154

	Agent s NN 21	Náhodný agent 1	Náhodný agent 2
Vítězství	475	8	17

	Agent s NN 21	Agent s NN -1	Agent s NN -1
Vítězství	412	49	39

	Agent s NN 21	Agent s NN 01	Agent s NN 01
Vítězství	211	140	149

	Agent s NN 21	Agent s NN 11	Agent s NN 11
Vítězství	302	106	92

	Agent s NN 31	Náhodný agent 1	Náhodný agent 2
Vítězství	472	19	9

	Agent s NN 31	Agent s NN -1	Agent s NN -1
Vítězství	366	59	75

	Agent s NN 31	Agent s NN 01	Agent s NN 01
Vítězství	218	152	130

	Agent s NN 31	Agent s NN 11	Agent s NN 11
Vítězství	245	119	136

	Agent s NN 31	Agent s NN 21	Agent s NN 21
Vítězství	177	151	172

	Agent s NN 02	Náhodný agent 1	Náhodný agent 2
Vítězství	462	20	18

	Agent s NN 02	Agent s NN -1	Agent s NN -1
Vítězství	440	27	37

	Agent s NN 12	Náhodný agent 1	Náhodný agent 2
Vítězství	469	22	9

	Agent s NN 12	Agent s NN -1	Agent s NN -1
Vítězství	370	42	88

	Agent s NN 12	Agent s NN 02	Agent s NN 02
Vítězství	302	102	96

	Agent s NN 22	Náhodný agent 1	Náhodný agent 2
Vítězství	464	17	19

	Agent s NN 22	Agent s NN -1	Agent s NN -1
Vítězství	411	56	33

	Agent s NN 22	Agent s NN 02	Agent s NN 02
Vítězství	125	191	184

	Agent s NN 22	Agent s NN 12	Agent s NN 12
Vítězství	307	109	84

	Agent s NN 32	Náhodný agent 1	Náhodný agent 2
Vítězství	460	15	25

	Agent s NN 32	Agent s NN -1	Agent s NN -1
Vítězství	373	50	77

	Agent s NN 32	Agent s NN 02	Agent s NN 02
Vítězství	207	154	140

	Agent s NN 32	Agent s NN 12	Agent s NN 12
Vítězství	252	119	129

	Agent s NN 32	Agent s NN 22	Agent s NN 22
Vítězství	254	111	135

	Agent s NN 01	Agent s NN 02	Agent s NN 02
Vítězství	175	150	175

	Agent s NN 02	Agent s NN 01	Agent s NN 01
Vítězství	275	118	107

	Agent s NN 11	Agent s NN 12	Agent s NN 12
Vítězství	225	150	125

	Agent s NN 12	Agent s NN 11	Agent s NN 11
Vítězství	232	155	113

	Agent s NN 21	Agent s NN 22	Agent s NN 22
Vítězství	267	115	118

	Agent s NN 22	Agent s NN 21	Agent s NN 21
Vítězství	165	188	147

	Agent s NN 31	Agent s NN 32	Agent s NN 32
Vítězství	226	127	147

	Agent s NN 32	Agent s NN 31	Agent s NN 31
Vítězství	160	172	168

Neuronové sítě s komplexnější topologií

	Agent s CNN -1	Náhodný agent 1	Náhodný agent 2
Vítězství	471	13	16

	Agent s CNN 01	Náhodný agent 1	Náhodný agent 2
Vítězství	448	30	22

	Agent s CNN 01	Agent s CNN -1	Agent s CNN -1
Vítězství	469	12	19

	Agent s CNN 11	Náhodný agent 1	Náhodný agent 2
Vítězství	412	45	43

	Agent s CNN 11	Agent s CNN -1	Agent s CNN -1
Vítězství	461	21	18

	Agent s CNN 11	Agent s CNN 01	Agent s CNN 01
Vítězství	211	150	139

	Agent s CNN 21	Náhodný agent 1	Náhodný agent 2
Vítězství	452	31	17

	Agent s CNN 21	Agent s CNN -1	Agent s CNN -1
Vítězství	457	24	19

	Agent s CNN 21	Agent s CNN 01	Agent s CNN 01
Vítězství	224	131	145

	Agent s CNN 21	Agent s CNN 11	Agent s CNN 11
Vítězství	249	122	129

	Agent s CNN 31	Náhodný agent 1	Náhodný agent 2
Vítězství	442	27	31

	Agent s CNN 31	Agent s CNN -1	Agent s CNN -1
Vítězství	460	17	23

	Agent s CNN 31	Agent s CNN 01	Agent s CNN 01
Vítězství	209	145	146

	Agent s CNN 31	Agent s CNN 11	Agent s CNN 11
Vítězství	201	145	154

	Agent s CNN 31	Agent s CNN 21	Agent s CNN 21
Vítězství	206	146	148

	Agent s CNN 02	Náhodný agent 1	Náhodný agent 2
Vítězství	413	52	35

	Agent s CNN 02	Agent s CNN -1	Agent s CNN -1
Vítězství	470	14	16

	Agent s CNN 12	Náhodný agent 1	Náhodný agent 2
Vítězství	445	27	28

	Agent s CNN 12	Agent s CNN -1	Agent s CNN -1
Vítězství	462	12	26

	Agent s CNN 12	Agent s CNN 02	Agent s CNN 02
Vítězství	250	124	126

	Agent s CNN 22	Náhodný agent 1	Náhodný agent 2
Vítězství	433	31	36

	Agent s CNN 22	Agent s CNN -1	Agent s CNN -1
Vítězství	468	18	14

	Agent s CNN 22	Agent s CNN 02	Agent s CNN 02
Vítězství	221	140	139

	Agent s CNN 22	Agent s CNN 12	Agent s CNN 12
Vítězství	271	117	112

	Agent s CNN 32	Náhodný agent 1	Náhodný agent 2
Vítězství	461	25	14

	Agent s CNN 32	Agent s CNN -1	Agent s CNN -1
Vítězství	475	12	13

	Agent s CNN 32	Agent s CNN 02	Agent s CNN 02
Vítězství	246	126	128

	Agent s CNN 32	Agent s CNN 12	Agent s CNN 12
Vítězství	294	98	108

	Agent s CNN 32	Agent s CNN 22	Agent s CNN 22
Vítězství	238	138	124

	Agent s CNN 01	Agent s CNN 02	Agent s CNN 02
Vítězství	175	150	175

	Agent s CNN 02	Agent s CNN 01	Agent s CNN 01
Vítězství	275	118	107

	Agent s CNN 11	Agent s CNN 12	Agent s CNN 12
Vítězství	225	150	125

	Agent s CNN 12	Agent s CNN 11	Agent s CNN 11
Vítězství	232	155	113

	Agent s CNN 21	Agent s CNN 22	Agent s CNN 22
Vítězství	267	115	118

	Agent s CNN 22	Agent s CNN 21	Agent s CNN 21
Vítězství	165	188	147

	Agent s CNN 31	Agent s CNN 32	Agent s CNN 32
Vítězství	226	127	147

	Agent s CNN 32	Agent s CNN 31	Agent s CNN 31
Vítězství	160	172	168

Jednoduchá a komplexnější topologie

	Agent s NN 01	Agent s CNN 01	Agent s CNN 01
Vítězství	210	135	155

	Agent s CNN 01	Agent s NN 01	Agent s NN 01
Vítězství	190	157	153

	Agent s NN 11	Agent s CNN 11	Agent s CNN 11
Vítězství	199	151	150

	Agent s CNN 11	Agent s NN 11	Agent s NN 11
Vítězství	244	135	121

	Agent s NN 21	Agent s CNN 21	Agent s CNN 21
Vítězství	203	144	153

	Agent s CNN 21	Agent s NN 21	Agent s NN 21
Vítězství	163	171	166

	Agent s NN 31	Agent s CNN 31	Agent s CNN 31
Vítězství	184	158	158

	Agent s CNN 31	Agent s NN 31	Agent s NN 31
Vítězství	166	171	163

	Agent s NN 02	Agent s CNN 02	Agent s CNN 02
Vítězství	236	130	134

	Agent s CNN 02	Agent s NN 02	Agent s NN 02
Vítězství	150	164	186

	Agent s NN 12	Agent s CNN 12	Agent s CNN 12
Vítězství	218	139	143

	Agent s CNN 12	Agent s NN 12	Agent s NN 12
Vítězství	235	134	131

	Agent s NN 22	Agent s CNN 22	Agent s CNN 22
Vítězství	190	157	153

	Agent s CNN 22	Agent s NN 22	Agent s NN 22
Vítězství	278	114	108

	Agent s NN 32	Agent s CNN 32	Agent s CNN 32
Vítězství	197	163	140

	Agent s CNN 32	Agent s NN 32	Agent s NN 32
Vítězství	283	115	102

	Agent s NN 01	Agent s CNN 02	Agent s CNN 02
Vítězství	206	147	147

	Agent s CNN 02	Agent s NN 01	Agent s NN 01
Vítězství	184	148	168

	Agent s NN 02	Agent s CNN 01	Agent s CNN 01
Vítězství	231	126	143

	Agent s CNN 01	Agent s NN 02	Agent s NN 02
Vítězství	187	159	154

	Agent s NN 11	Agent s CNN 12	Agent s CNN 12
Vítězství	218	152	130

	Agent s CNN 12	Agent s NN 11	Agent s NN 11
Vítězství	240	135	125

	Agent s NN 12	Agent s CNN 11	Agent s CNN 11
Vítězství	217	139	144

	Agent s CNN 11	Agent s NN 12	Agent s NN 12
Vítězství	244	124	132

	Agent s NN 21	Agent s CNN 22	Agent s CNN 22
Vítězství	207	156	137

	Agent s CNN 22	Agent s NN 21	Agent s NN 21
Vítězství	188	135	177

	Agent s NN 22	Agent s CNN 21	Agent s CNN 21
Vítězství	184	153	163

	Agent s CNN 21	Agent s NN 22	Agent s NN 22
Vítězství	223	143	134

	Agent s NN 31	Agent s CNN 32	Agent s CNN 32
Vítězství	150	230	120

	Agent s CNN 32	Agent s NN 31	Agent s NN 31
Vítězství	241	141	118

	Agent s NN 32	Agent s CNN 31	Agent s CNN 31
Vítězství	178	174	148

	Agent s CNN 31	Agent s NN 32	Agent s NN 32
Vítězství	196	146	158

Celkové výsledky

	MCTS agent	Náhodný agent 1	Náhodný agent 2
Vítězství	96	0	4

	MCTS agent s NN	Náhodný agent 1	Náhodný agent 2
Vítězství	92	6	2

	MCTS agent	MCTS agent s NN	Agent s NN
Vítězství	45	35	20

C. Příloha - Obsah CD

Obecný obsah:

- RiskAI.pdf - tato práce v elektronické podobě
- složka doc obsahuje programátorskou dokumentaci, uživatelskou dokumentaci a vygenerovanou html dokumentaci
- složka build obsahuje zkompilovanou solution hry Risk
- složka hra_Risk obsahuje všechny zdrojové soubory. Najdeme zde projekty AI, AIvsAI, RiskModel, RiskViewModel, RiskNetworking, Risk a Server.
- složka AI obsahuje serializované neuronové sítě
- v případě kompilace zdrojových souborů je připravená solution hra_Risk/Risk/Risk.sln

Seznam binárních souborů:

- build/Server.exe - server pro hru Risk
- build/Risk.exe - klient pro hru Risk
- build/AIvsAI.exe - nástroj pro testování a učení agentů