Návrh a implementace „AI“ hráče pro hru Yahtzee s využitím rozhodovacího stromu Expectimax

Úvod

Nedávno mi kamarád ukázal velmi zajímavou hru, o které jsem paradoxně nikdy neslyšel. Jmenuje se Yatzy nebo Yahtzee a je původně kanadská. Hra se podle názvu lehce liší v pravidlech. Hra mě velice zaujala, protože se zde spojuje náhoda (hod kostkami) se strategií (které kostky držet a které přehodit). Jinak řečeno, musíte mít trochu štěstí, ale i když ho člověk nemá, může dost bodů získat správnou strategií.

Hráč má cíl nahrát v 13 kolech co nejvíce bodů. Každý tah hráče povoluje hod pěti kostkami a následné dva přehozy. A teď ta zajímavá část: můžeme si vždy vybrat, které kostky budeme přehazovat a které držet. Po celkem třech hodech v jednom tahu tedy mají kostky finální hodnoty a z těch se pak vždy musí zapsat hodnota do jedné ze 13 bodovaných kategorií skórovací karty. Hráč si může vybrat jakoukoli kategorii, což dává další prostor ke strategii.

Dal jsem si tedy za cíl naprogramovat mechaniku hry s jedním klíčovým prvkem – rozhodovacím algoritmem. Algoritmus je postavený na principu Minimax algoritmu, ale je upravený pro hru s náhodou – v praxi uváděn jako Expectimax algoritmus.

Zadání chce vlastní implementaci jedné z pokročilých datových struktur. Já tedy budu implementovat algoritmus Expectimax, který dynamicky generuje a prochází rozhodovací strom hry. To bude jádro počítačového („AI“) hráče, který se bude pokoušet za 13 kol dosáhnout co nejvyššího skóre. Pro implementaci volím jazyk Python.

Expectimax strom sice bude jádrem autonomního hráče, ale celý proces se bude ještě spoléhat na lidské heuristiky. Heuristická funkce bude ve finále to, co ohodnotí koncové stavy (listy stromu). Roli zde budou hrát různé penalizace a odměny. Samotný algoritmus pak tato ohodnocení zprůměruje a vybere optimální rozhodnutí s nejvyšší očekávanou hodnotou.

Výsledkem bude plně funkční hra v CLI. Hra bude možná hrát v manuálním módu bez asistence, potom v plně autonomním módu kdy „AI“ hráč odehraje 13 kol a bude průběžně vypisovat rozhodnutí včetně skórovací karty, dále v manuálním módu s asistencí, kdy nám „AI“ hráč radí optimální rozhodnutí držení kostek a konečného zápisu do kategorie. Čtvrtý mód bude pouze benchmark, kdy necháme autonomní mód odehrát N her a spočítáme statistiky přes těch N her a zjistíme, jak je náš hráč dobrý.

Teoretický popis

Pravidla hry

Níže uvedu přesná pravidla pro moji implementaci hry. Měla by to být přesná pravidla hry s názvem Yahtzee, ale na internetu se to často motá [1]. Hráč má na každý tah právě jeden počáteční hod a dva přehozy, při kterých může držet a přehazovat libovolné kostky. Kostky, které při prvním přehozu držel nemusí držet při druhém, může libovolně přehodit držené kostky nebo držet přehozené kostky. Jedinou podmínkou je, že nesmí překročit počet přehozů 2. Hráč může kdykoli (když je s hodem spokojený) ukončit hod – vlastně řekne: „Chci držet všechny kostky a zapsat."

Všech 13 kategorií skórovací karty, které jsou na začátku prázdné musí být po 13 kole vyplněny. To znamená, že po každém tahu musíme něco zapsat, neexistuje tah bez zapsání. Ke kategoriím je velmi důležité dodat, že mohou být i „proškrtnuty" (obětovány se skóre 0). Kategorie jsou:

- **Horní sekce (Bonus:** Pokud je součet skóre horní sekce 63 a víc, hráči se přidá bonus ve výši 35 bodů**)**
    - **Jedničky:** Kostky s hodnotou 1 (bodová hodnota: součet všech kostek s hodnotou 1)
    - **Dvojky:** Kostky s hodnotou 2 (bodová hodnota: součet všech kostek s hodnotou 2)
    - **Trojice:** Kostky s hodnotou 3 (bodová hodnota: součet všech trojek)
    - **Čtyřky:** Kostky s hodnotou 4 (bodová hodnota: součet všech kostek s hodnotou 4)
    - **Pětky:** Kostky s hodnotou 5 (bodová hodnota: součet všech kostek s hodnotou 5)
    - **Šestky:** Kostky s hodnotou 6 (bodová hodnota: součet všech kostek s hodnotou 6)
- **Dolní sekce**
    - **3 stejné:** Tři stejné kostky (bodová hodnota: součet všech kostek)
    - **4 stejné:** Čtyři stejné kostky (bodová hodnota: součet všech kostek)
    - **Full House:** Dvojice stejných kostek a trojice stejných kostek (bodová hodnota: 25 bodů)
    - **Malá postupka:** 1, 2, 3 a 4; 2, 3, 4 a 5; nebo 3, 4, 5 a 6 (bodová hodnota: 30 bodů)
    - **Velká postupka:** 1, 2, 3, 4 a 5 nebo 2, 3, 4, 5 a 6 (bodová hodnota: 40 bodů)
    - **Yahtzee:** Pět stejných kostek (bodová hodnota: 50 bodů)
    - **Šance:** Jakákoli kombinace 5 kostek (bodová hodnota: součet všech kostek)

Po 13 kolech se sečte celkové skóre horní a dolní sekce, v horní sekci se přidává 35 bodů, pokud je součet 63 a více.

Expectimax algoritmus

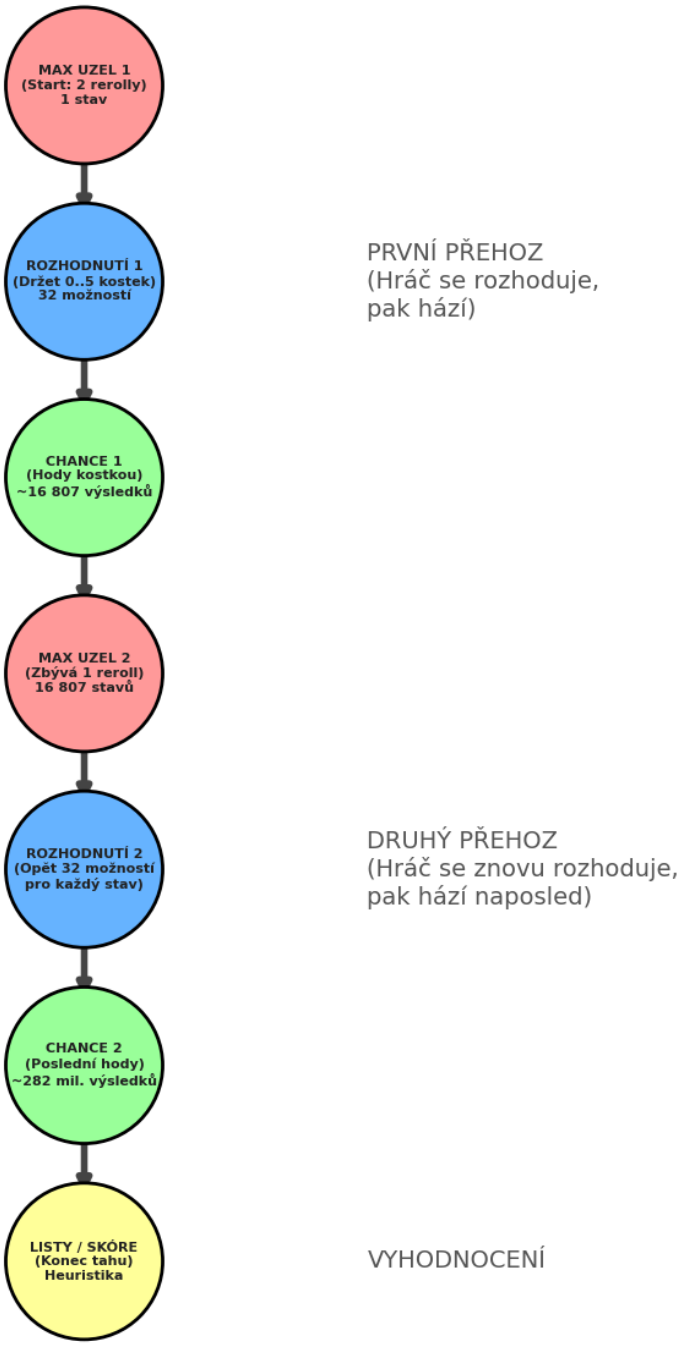
Pro pochopení algoritmu Expectimax je vhodné začít u jeho předchůdce – algoritmu Minimax. Ten se využívá v deterministických hrách s nulovým součtem (např. šachy), kde proti sobě stojí dva soupeři. Hráč 1 (MAX) se snaží maximalizovat své skóre, zatímco Hráč 2 (MIN) se snaží skóre minimalizovat. Příkladem může být šachový engine Stockfish, který využívá vysoce optimalizovaný Minimax strom s technikami jako Alpha-Beta prořezávání (pruning) a moderně i neuronové sítě pro evaluaci pozic [2].

V kostkové hře Yahtzee však proti nám nestojí inteligentní oponent, který by aktivně ovlivňoval výsledek hodu v náš neprospěch. Místo hráče MIN zde vstupuje do hry faktor náhody. Kostky nevolí nejhorší možný výsledek, ale padají podle statistického rozdělení pravděpodobnosti.

Proto klasický Minimax pro Yahtzee nelze použít. Místo minimalizačních uzlů zavádíme tzv. CHANCE (náhodné) uzly. Tyto uzly reprezentují moment hodu kostkami. Cílem algoritmu v tomto bodě není najít minimum, ale vypočítat očekávanou hodnotu (Expected Value) – tedy vážený průměr všech možných výsledků hodu, vážený jejich pravděpodobností. Výsledný algoritmus se proto nazývá

Expectimax. Střídají se v něm dva typy uzlů. MAX uzly, kde se hráč rozhoduje, které kostky podržet, abychom maximalizovali zisk. A CHANCE uzly, kde simulujeme hod kostkami a počítáme průměrný zisk ze všech možných výsledků. [3,4]

Pro naši hru to v praxi znamená, že na začátku bude MAX_1 uzel s prvním hodem, z něj půjde 32 (2^5) možností držení kostek. Z každého držení kostek půjde CHANCE_1 uzel, který provede takový počet simulací, aby to odpovídalo počtu přehazovaných kostek *n_reroll* tzn. 6^{n_reroll}. Celkově je pak v další vrstvě až 16807 výsledků. Tyto výsledky dále každý putují do svého MAX_2 uzlu, z něj opět půjde 32 (2^5) možností držení kostek. Z každého držení kostek půjde CHANCE_2 uzel, který znovu provede odpovídající počet simulací. Z těchto CHANCE_2 už poté vycházejí listy (terminály), celkem jich v nejhorším případě může být ~282 milionů. Pro lepší pochopení je níže obrázek č. 1.



Obrázek č. 1: Struktura tahu a počty stavů

Po tom, co máme koncové skóre zavoláme heuristiky, které vyhodnotí na kolik bodů si list na základě fáze hry a dalších kritérií cení. Tyto body potom dostane CHANCE_2 a udělá průměr všech svých dětí, což je očekávaná hodnota (expected value). MAX_2 potom vybere ze svých 32 možných držení tu s nejvyšší očekávanou hodnotou. Takhle se ta nejlepší očekávaná hodnota zpětně dopropaguje až k našemu prvnímu hodu a máme rozhodnutí co držet. Listů sice nebude většinou 282 milionů, ale i tak jich bude astronomické číslo a program by zamrzl. Musíme memorizovat a optimalizovat. [3,4]

Strategie hry a heuristiky

Yahtzee je sice hra založená na náhodě, ale dlouhodobý úspěch v ní závisí na schopnosti efektivně řídit riziko a maximalizovat pravděpodobnost budoucích zisků. Přístup ke hře se nesoustředí pouze na aktuální hod, ale vnímá každé rozhodnutí v kontextu celé partie. To budou dělat naše heuristické funkce. Expectimax strom jen následně spočítá co držet, ale ta skóre listů jako takové přidělí heuristika. Dobrá strategie musí vyvažovat okamžitý bodový zisk s udržováním otevřených možností pro další tahy.

Základním část dobré taktiky je útok na horní bonus. Získání dodatečných 35 bodů za dosažení součtu 63 v horní sekci dost navýší skóre. Zkušený hráč proto upřednostňuje zápis do horní sekce, i když by mu zápis do dolní sekce přinesl o pár bodů více. Klíčové je zde především neplýtvat vyšší čísla jako pětky a šestky. Zapsat si do kolonky šestek pouze jednu nebo dvě kostky je strategická chyba, která se později těžko dohání.

Druhým klíčovým aspektem je management záchranných sítí především kategorie šance. Začátečníci často dělají chybu, že šanci využijí příliš brzy na průměrný hod (např. 20 bodů). Strategicky správné je však držet šanci déle jako pojistku pro situace, kdy v pozdní fázi hry nepadne žádná kýžená kombinace. Pokud hráč vyplýtvá šanci v pátém kole, vystavuje se v koncovce obrovskému riziku nuceného proškrtávání hodnotných kategorií. Na druhou stranu má šance potenciál být až 30, takže si zachránit nějaký velmi podprůměrný hod nemusí být také nejlepší.

Se záchrannými sítěmi souvisí taktika "proškrtávání" (obětování). V situaci, kdy hod absolutně neodpovídá žádné volné kolonce, musí hráč zvolit "nejmenší zlo". Obecně platí pravidlo obětovat to, co má nejmenší dopad na celkové skóre a bonus. Typicky se jako první obětují jedničky, protože ztráta maximálně 5 bodů je zanedbatelná. Naopak proškrtnout si velkou postupku za 40 bodů je až poslední zoufalá možnost.

V neposlední řadě se strategie mění v závislosti na fázi hry.

- V úvodu je výhodné hrát agresivně, riskovat a pokoušet se o těžké kombinace (Yahtzee, postupky), protože v případě neúspěchu je stále dostatek "levných" kategorií pro zápis.
- Ve střední fázi se hráč soustředí na konsolidaci bonusu a plnění dolní sekce.
- V koncovce se hra mění na čistou matematiku pravděpodobnosti, kde je nutné hrát defenzivně a minimalizovat ztráty, protože možnosti nápravy chyb jsou vyčerpány.

Všechny tyto principy se budou musí ve skriptu heuristik uplatnit, aby byl bot kvalitní. Může být agresivnější a čekat na Yahtzee až do konce s potenciálem vyššího skóre (i přes 300), ale i velkého selhání (pod 180). Nebo může být defenzivnější a častěji dosáhnout rozumného skóre přes 200 za cenu toho, že méněkrát dosáhne opravdu vysokého skóre. Uvidíme, která možnost dá při velkém počtu her lepší průměrné skóre.

Popis vytvořeného SW

Předem bych rád podotkl, že i když jsem se snažil, narážely mé programátorské schopnosti na tomto (pro mě docela složitém) projektu na strop a rozhodně bych z nuly nevěděl, kde začít. Proto se rovnou přiznám, že jsem řešení, a především architekturu celého projektu hluboce konzultoval s LLM [5]. Často mě přivedl na velmi zajímavá a užitečná zjednodušení, která jsem převzal a pomohla mi výrazně optimalizovat kód. Snažil jsem se však vyhnout „vibe-codingu“ (slepému kopírování) a porozumět každému řádku – možná i proto mi práce trvala déle a odevzdávám ji až v lednu.

Jednou z nejužitečnějších technik, kterou jsem si díky konzultacím osvojil, je využití bitových operací pro reprezentaci stavu hry. Konkrétně operátor bitového posunu „<<“ umožňuje efektivně pracovat s kombinacemi. V našem případě každé číslo (1-32) odpovídá jedné z 2^5 kombinací pro držení pěti kostek. Například číslo 5 (binárně 00101) díky bitové logice okamžitě reprezentuje stav "držím první a třetí kostku". Tato reprezentace je mnohem rychlejší a paměťově úspornější než práce se seznamy. Ještě se mi moc líbilo, že mě LLM naučil dělat typové anotace funkcí a proměnných, což mi přijde super a extrémně přehledné. Taky jsem se díky LLM dost dobře snažil dodržovat private a protetcted metody uvnitř tříd.

Kromě toho mi LLM výrazně pomohl s návrhem struktury heuristik, což mi umožnilo iterativně ladit herní konstanty a sledovat dopad na průměrné skóre. Celkově jsem veškerý kód (včetně generovaných částí) plně pochopil a okomentoval, aby byla logika z jednotlivých skriptů zřejmá.

Architektura

Architektura projektu byla zvolena plně modulární, rozdělená na skripty s hlavním *main.py*, do kterého se vše naimportováno ze zbývajících skriptů. Tento skript je ten, co spustíme a kde máme na výběr mód hry 1-4.

- *constants.py* – konstanty hry, ne všechny jsou nutné, ale pro přehlednost v kódu jsou super
- *scoring.py* – stará se o bodování kategorií a ověřuje, zda je kategorie splněna
- *scorecard.py* – Obsahuje třídu ScoreCard, která reprezentuje stav skórovací karty. Kvůli rekurzivním výpočtům ve stromu je navržena jako immutable, zároveň hlídá obsazenost kategorií a nárok na horní bonus.
- *heuristics.py* – asi zdaleka nejsložitější skript, implementuje funkce, které na základě aktuálního hodu, fáze hry a stavu scorecard vybírají optimální kategorii pro zápis.
- *expectimax_turn.py* – pro splnění projektu nejdůležitější, implementuje algoritmus Expectimax, který prohledává stavový prostor hry (strom rozhodnutí) a vrací optimální strategii pro držení kostek a přehazování.
- *players.py* – zde řeším pěkné formátování kostek, zavádím třídu *Player,* ze které pak dědí *HumanPlayer* i *AIPlayer.* U HumanPlayer se musí vyřešit inputy.
- *game.py* – výborná pro přehlednost, jen zabalí hru do funkce. Přijímá jako argumenty typ hráče, zda vykreslovat průběh a vrací skórovací kartu vyplněnou.
- *benchmark.py* – funkce pro běh mnoha her a vyhodnocení statistik

Skripty tvořící „AI“ hráče

Níže budou podrobněji popsány skripty tvořící jádro naší implementace „AI“ hráče. Kód by jinak měl být dostatečně okomentovaný, takže nepůjdu řádek po řádku.

Heuristics.py

Tento modul slouží k ohodnocení listových stavů ve stromu, tedy k rozhodnutí, kterou kategorii zvolit nejen pro okamžitý bodový zisk, ale s ohledem na dlouhodobou strategii. Na začátku souboru jsou definovány ladicí konstanty, které formují herní styl AI. Určují například, jak agresivně usilovat o horní bonus, jak přísně penalizovat slabé zápisy do Chance v rané fázi hry nebo jak chránit vysoké horní kategorie před nevýhodným zaplněním. Tyto váhy jsou klíčové, protože algoritmus Expectimax nepropočítává hru až do konce, ale maximalizuje hodnotu vrácenou právě touto heuristikou. Tato hodnota proto musí co nejpřesněji odrážet skutečný potenciál daného stavu.

Velmi podstatná část heuristik se věnuje práci s horním bonusem. Místo pevné hodnoty 35 bodů se bonus přepočítává na očekávanou hodnotu podle toho, jak realistické je jej ještě dosáhnout. Funkce *_bonus_probability(upper_sum, remaining_upper)* vrací odhad pravděpodobnosti v rozmezí 0 až 1, kde porovnává kolik bodů ještě chybí do limitu 63 s maximálním možným ziskem ze zbývajících horních kategorií. Výsledný poměr je navíc umocněn na 1.7, což vytváří pesimističtější chování AI ohledně bonusu. Na tuto funkci navazuje *_bonus_ev(scorecard)*, která jednoduše vrací součin pravděpodobnosti a hodnoty bonusu (p × 35). Očekávaná hodnota tak může být nižší než 35, což nám pěkně usnadní rozhodování, jak moc je reálný.

Nejdůležitější funkce, která dělá nejvíc práce je *evaluate_write(dice, scorecard, cat) -> float*, která vrací „spokojenost“ s hypotetickým zápisem *cat* pro aktuální kostky. Základ tvoří okamžité body *scorecategory(dice, cat)*, ale na to se nabalují strategické penalizace a bonusy závislé na fázi hry (*turn_sleft/13*) a na dopadu do upper části. Klíčový blok je změna očekávané hodnoty bonusu. U upper kategorií se spočítá očekávaná hodnota bonusu před zápisem a po zápisu. To se odečte a tento rozdíl se násobí konstantou BONUS_PRESSURE. Další část heuristik se zaměřuje na kategorii Chance. V rané a střední fázi hry je Chance považována za cennou „záchrannou brzdu“ pro nepovedené hody. Heuristika ji proto přímo nezakazuje, ale odrazuje penalizací. Penalizuje ji ale jen tehdy, pokud by zapsaný součet nedosahoval stanovených cílových hranic (*CHANCE_TARGET_EARLY / CHANCE_TARGET_MID*). Systém dále postihuje nevýhodné zaplnění vysokých horních kategorií (zejména čtyřek, pětek a šestek) nízkými body. Tím chrání potenciál pro získání bonusu i vysokého celkového skóre. Podobně jsou penalizovány i slabé zápisy do kategorií 3-kind a 4-kind v době, kdy do konce hry zbývá ještě mnoho kol. Významnou roli hraje také strategie záměrného škrtání (zápisu nuly). Vzácné kategorie jako Straight, Full House či Yahtzee mají v rané fázi nastavenou vysokou penalizaci za proškrtnutí, později už nižší. U horních kategorií se penalizace škáluje podle hodnoty kostky – škrtnutí šestek (Sixes) je strategicky mnohem nákladnější než škrtnutí jedniček (Ones).

Celý proces hodnocení se uzavírá ve funkci *choose_bestcategory(dice, scorecard)*, která projde všechny volné kategorie, pro každou vypočítá skóre pomocí *evaluate_write()* a vybere tu s nejvyšší hodnotou. Právě tuto funkci využívá algoritmus Expectimax v listových uzlech stromu. Stejně tak slouží i samotným třídám hráčů (AI nebo člověku s nápovědou) pro finální rozhodnutí o zápisu na konci tahu.

Expectimax_turn.py

Klíčovým prvkem pro reprezentaci hráčovy volby je hold_mask – 5bitová maska (rozsah 0–31), kde každý bit odpovídá jedné kostce a určuje, zda ji hráč podrží (1) nebo přehodí (0). Díky tomuto přístupu je generování všech možných tahů v MAX uzlu triviální operací, která spočívá v iteraci přes 32 možných masek. Aby bylo možné efektivně využívat memoizaci, stav kostek se ve stromu vždy ukládá jako seřazený *tuple(sorted(dice))* čímž se eliminuje vliv pořadí kostek na vyhledávání v cache (např. hod 1-2-3 je identický jako 3-2-1).

Zásadní roli pro výkon hraje LRU cache *(@lru_cache(maxsize=300000))*, použitá u funkcí uzlů *_max_node()* a *_chance_node()*. Ve stromu se totiž velmi často opakují identické stavy (stejná kombinace kostek, stejný počet zbývajících hodů, stejný stav skóre). Bez cache by se tyto podúlohy

počítaly opakovaně. To by vedlo k exponenciálnímu nárůstu výpočetního času, zejména v CHANCE uzlech, kde se sčítá 6^n možností pro n přehazovaných kostek. Pro další urychlení si modul předem vypočítává *OUTCOMES[n]* pomocí *itertools.product()* pro všechna n od 1 do 5. Kombinace možných hodů jsou tak okamžitě k dispozici bez nutnosti jejich opakovaného generování.

Výsledek rozhodovacího procesu zapouzdřuje třída *Recommendation*, která obsahuje nejlepší *hold_mask*, volitelný tip na cílovou kategorii (*targe_tcategory*) a především *expected_value*, tedy celkovou očekávanou hodnotu tahu. Metody *held_indices()* a *reroll_indices()* slouží pouze k převodu bitové masky na srozumitelný seznam indexů kostek pro výpis nebo pro radu lidskému hráči. Vstupním bodem pro AI je funkce *recommendaction(dice, rerollsleft, scorecard)*, která zanalyzuje situaci a vrátí doporučení. Pokud už hráči nezbývají žádné rerolly, funkce rovnou přejde k výběru nejlepší kategorie pomocí heuristik.

Samotný výpočet probíhá v *_max_node(dice, rerolls_left, scorecard),* který pro každou z 32 masek určí její očekávanou hodnotu – buď přímo přes heuristiku (pokud se nic nepřehazuje), nebo voláním *_chance_node()*. Následný *_chance_node(held_dice, n_reroll, reroll_sleft, scorecard)* zprůměruje hodnoty všech možných výsledků hodu z *OUTCOMES* a volá další *_max_node(),* nebo (pokud už došly rerolly) ukončí větev heuristickým vyhodnocením. Doplňkovou funkcí je *guess_target_category(.)*, která se snaží odhadnout hráčův záměr (např. držení čtyř stejných kostek naznačuje pokus o Yahtzee), ale není důležitá a může vrátit i None. Pro správu paměti je k dispozici funkce *clear_ai_cache()*, která maže cache obou typů uzlů, což je užitečné například mezi jednotlivými hrami v benchmarku.

Skripty nutné pro funkčnost hry

Constansts.py
Obsahuje herní konstanty (počet kostek, počet stran, počet hodů v tahu) a hlavně class *Categor*y jako *IntEnum*, což je důležité, protože kategorie jsou zároveň pojmenované a zároveň fungují jako čísla. To se hodí pro indexaci do scores tuple i pro bitové pozice ve filled_mask. Soubor také rozděluje kategorie na UPPER_CATEGORIES a LOWER_CATEGORIES, což dělá především kvůli jednodušší prací s bonusem a jeho trackováním. Také drží CATEGORY_NAMES pro čitelné UI výpisy.

Scoring.py
Je to jen bodování podle pravidel. *Countfaces(dice)* vrátí četnosti hodnot 1-6. Další metoda je potom s*corecategory(dice, cat)*, která z četnosti spočítá body pro konkrétní kategorii. Zajímavý detail je kontrola small straight přes *sorted(set(dice)),* kde set odstraní duplicity a výrazně zjednoduší hledání souvislé sekvence. Datový typ *set* jsem předtím neznal.

Scorecard.py
Karta skóre (*ScoreCard*) definuje datový model stavu hry, tedy které kategorie už jsou vyplněné, jaké body jsou v nich zapsané a kolik bodů má hráč aktuálně v horní části kvůli bonusu. Nejdůležitější designová volba je použití *@dataclass(frozen=True, slots=True)*. *Frozen=True* zajišťuje, že objekt je immutable a *slots=True* zrychluje přístup k atributům, což se hodí, protože se *scorecard* často používá v AIPlayer. Immutabilita má praktický dopad, protože místo „patchování“ stavu se po každém zápisu vytváří nová instance. To je to bezpečné pro cachování v algoritmech, které si ukládají výsledky pro stejné vstupní stavy

Uvnitř *scorecard* se pro jednoduchost používá bitmaska *filled_mask*, kde každý bit reprezentuje jednu kategorii. Test „je kategorie vyplněná?“ se dá udělat jedním bitovým AND v metodě *is_filled()*. Skutečné bodové hodnoty jsou uložené v *scores*, což je *tuple[Optional[int], ...],* kde *None* znamená „ještě nezapsáno“ a číslo znamená bodový zápis v dané kategorii. Takto je to děláno kvůli nulám, které by tam pak dělaly nepořádek.

Aby se *scores* správně inicializovalo pro každou novou hru, používá se *field(default_factory=...)*, konkrétně *lambda* funkce, která vyrobí nový *tuple* délky *len(ALL_CATEGORIES)* vyplněný hodnotami *None*. Tohle je extra důležité, protože tím se garantuje, že při každém vytvoření *ScoreCard()* vznikne čerstvá výchozí struktura odpovídající aktuálnímu počtu kategorií [6]. Nemusíme tím pádem riskovat žádné sdílení.

Nejdůležitější metodou je *with_score(cat, points)*, která realizuje update. Vezme *scores* (tuple), převede ho na list (mutable), zapíše body do indexu *cat*, porom nastaví příslušný bit ve *filled_mask* a nakonec vrátí úplně novou instanci *ScoreCard* s aktualizovanými daty. Současně se v této metodě průběžně udržuje *upper_sum*, aby šel rychle vyhodnotit bonus přes *upper_bonus()*. celkové skóre přes *total_score(),* bez nutnosti pokaždé procházet celou tabulku. Pro přehlednost výstupu má *ScoreCard* i přepsané *__str__*, které formátuje vše do pěkné tabulky pro terminál.

Players.py
Skript spojuje logiku tahu s interakcí (AI nebo člověk) a řeší i to, jak se kostky vypisují. Na začátku jsou pomocné funkce pro ASCII render kostek (*diceface()*, *formatdicevisual()*), kde se kostky skládají vedle sebe a nad ně se vytisknou indexy 0–4, aby uživatel mohl jednoduše napsat třeba „0 2 4“.

Základní návrh je, že *TurnResult* je malá třída, která nese *final_dice*, *chosen_category* a *score*, takže hra neřeší, kdo tah odehrál, jen zpracuje výsledek. *Player* je parent class, která definuje rozhraní *take_turn(scorecard)*, ale sama nic nedělá – proto nemá vlastní *__init__* a slouží hlavně k jednotnému typu pro *game.py*. Společné metody jsou *roll_dice(n)* pro hod n kostkami a *reroll_dice(dice, hold_mask)*, která podle bitmasky přehodí jen nedržené indexy a vrátí nový tuple.

AIPlayer implementuje *taket_turn* tak, že po prvním hodu opakovaně (max 2x) volá *recommendaction(dice, rerolls_left, scorecard)*, vypíše doporučení a provede reroll pomocí *reroll_dice()*, dokud buď nedojdou rerolly, nebo AI nedoporučí „drž vše“ (mask 31, bitově 11111). Na konci AI zvolí kategorii přes *choose_bestcategory()* a bodování dopočítá přes *scorecategory()*, takže výsledkem je vždy konzistentní TurnResult class.

HumanPlayer dělá podobný loop, ale místo výběru masky algoritmem čte vstupy od uživatele (prázdný vstup = držet vše, *all* = přehodit vše, jinak seznam indexů, oddělený mezerami). V režimu (*advisor=True*) před každým přehodem vypíše hint z AI (*rec.target_category()),* ta ale často píše nesmysl, kvůli fallbacku stromu. Na konci vypíše dostupné kategorie s tím, kolik by daly bodů pro aktuální hod, a uživatel vybere index kategorie. V režimu (*advisor=True*) se navíc vypíše i doporučená kategorie z *choose_best_category().*

Game.py
Definuje *play_game(player, printing=True, seed=None*), vytvoří prázdnou scorecard a 13x zavolá *player.take_turn(scorecard).*

Po každém tahu se stav aktualizuje přes scorecard = *scorecard.with_score(result.chosen_category, result.score)*. Na konci se vytiskne finální skóre a vrátí se finální *ScoreCard*, což pak používá i benchmark a main menu.

Závěr

V této zprávě jsem popsal svůj funkční projekt „AI“ hráče pro hru Yahtzee pomocí Expectimax algoritmu a heuristik. Při spuštění benchmarku na 1000 her jsem dosáhl těchto výsledků:

- **Průměrné skóre: 224.31**
- Medián: 219.00

- Směrodatná odchylka: 36.48
- Minimum: 128
- Maximum: 336
- Percentily:
    - 10 %: 182.0
    - 25 %: 199.0
    - 50 %: 219
    - 75 %: 253.0
    - 90 %: 269.0

Výsledky považuji za uspokojivé, neboť dosažené průměrné skóre se blíží teoretickému optimu 254,59 bodů, které uvádí odborná literatura. Domnívám se, že dalším jemným laděním heuristických vah by bylo možné se této hranici přiblížit ještě více.

Teď ještě dokončím myšlenku o kategorii Yahtzee, kterou jsem naťukl na začátku. Během testování jsem zjistil klíčovou informaci týkající se strategie pro kategorii Yahtzee. Ukázalo se, že v pozdní fázi hry je statisticky výhodnější tuto kategorii obětovat a zachránit tak body jinde, než doufat v nepravděpodobný zázrak. Původní verze AI používala fixní penalizaci *ZERO_RARE_BASE = 20.0* bez ohledu na fázi hry, což vedlo k průměrnému skóre pouze 221,18 bodu.

Výrazného zlepšení jsem dosáhl zavedením dynamické penalizace závislé na počtu zbývajících kol. V rané fázi (více než 7 kol do konce) je nyní penalizace za škrtnutí Yahtzee uměle navýšena (o 8.0 × phase), aby AI riskovala a pokoušela se o vysoký zisk. Naopak v pozdní fázi (méně než 5 kol) penalizace klesá na minimum, což umožňuje spálit Yahtzee na nechtěný hod a maximalizovat jistý zisk v jiných kategoriích.

Zdroje

[1] *Pravidla Yahtzee: Návod ke hře pro začátečníky*. Online. WikiHow. Dostupné z: https://www.wikihow.cz/Jak-yahtzee. [cit. 2026-01-04].

[2] *3.2 Minimax*. Online. Introduction to Artificial Intelligence. Dostupné z: https://inst.eecs.berkeley.edu/~cs188/textbook/games/minimax.html. [cit. 2026-01-04].

[3] *Expectimax Algorithm in Game Theory*. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/dsa/expectimax-algorithm-in-game-theory/. [cit. 2026-01-04].

[4] CONTRIBUTORS, Wikipedia. *Expectiminimax*. Online. In: Wikipedia, The Free Encyclopedia. Dostupné z: https://en.wikipedia.org/wiki/Expectiminimax. [cit. 2026-01-04].

[5] Google. (2024). Gemini 3 Pro [Large language model]. https://deepmind.google. [cit. 2026-01-04].

[6] *Python Trick: Using dataclasses with field(default_factory=...)*. Online. DEV Community. Dostupné z: https://dev.to/devasservice/python-trick-using-dataclasses-with-fielddefaultfactory-4159. [cit. 2026-01-04].

[7] *Built-in Types*. Online. Python documentation. Dostupné z: https://docs.python.org/3/library/stdtypes.html. [cit. 2026-01-04].

[8] *Functools — Higher-order functions and operations on callable objects*. Online. Python documentation. Dostupné z: https://docs.python.org/3/library/functools.html. [cit. 2026-01-04].