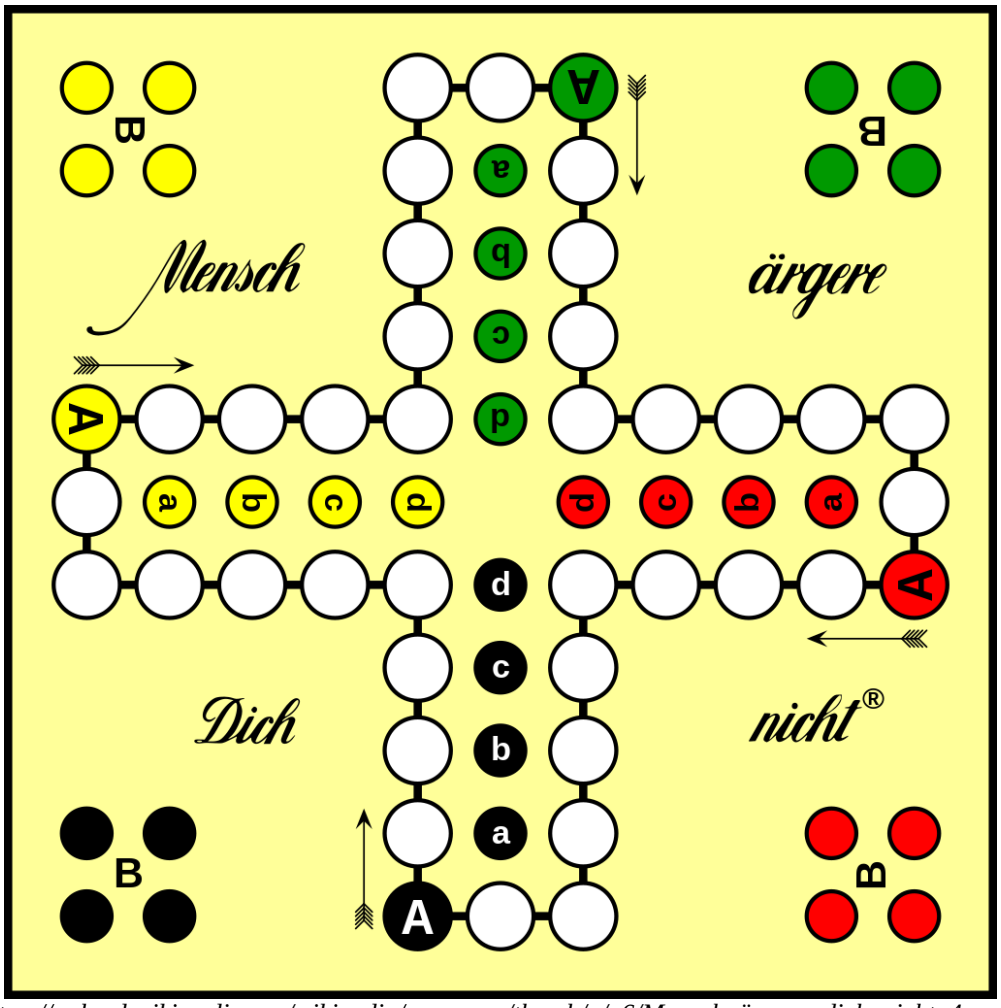
# Človeče nehnevaj sa

V treťom zadaní namodelujete obľúbenú spoločenskú hru, Človeče nehnevaj sa. Sústredíme sa pritom na štandardnú verziu hry so štyrmi hráčmi, kde cieľom je dostať figúrky do domčeku čo najskôr. Na začiatku hry má každý hráč všetky štyri figúrky v „stajni“ (polia označené B na obrázku nižšie). Hráči striedavo hádžu kockou, figúrku môžu postaviť na štartovacie pole (označené A na obrázku nižšie) ak padne šestka. Figúrky posúvajú po hracej ploche v smere hodinových ručičiek na základe hodu kocky, po každom hode 6 si môžu hod zopakovať, a posúvať figúrku o súčet padnutých hodov.




*Zdroj: https://upload.wikimedia.org/wikipedia/commons/thumb/a/a6/Mensch_ärgere_dich_nicht_4.svg/1024px-Mensch_ärgere_dich_nicht_4.svg.png*

Pri presune figúrky môžu nastať tri situácie:
- cieľové pole je prázdne, v tomto prípade figúrka stúpi na toto pole;
- na cieľovom poli stojí iná figúrka hráča, v tomto prípade je ťah neplatný a hráč musí posunúť inú figúrku (ak je to možné);
- na cieľovom poli stojí figúrka iného hráča, v tomto prípade hráč svoju figúrku posunie na pole, a súčasne vybije druhú figúrku, pošle ju naspäť do stajne, a nahnevá svojho protivníka na život.

Ak figúrka prešla jedno celé kolo po hracej ploche (od bodu A po pole napravo od štartovacej pozície), pôjde do domčeka (polia označené a–b na obrázku vyššie), kde už nemôže byť vybitá

protihráčmi. Figúrka sa dostane do domčeka iba v prípade, ak je daná pozícia voľná. Napríklad, ak figúrka stojí na poslednej pozícii pred domčekom a padne 1, no pozícia „a“ je už obsadená, musí si počkať na ďalšie kolo a veriť, že bude mať viac šťastia (a nikto ju medzitým nevybije).

Pravidlá hry sú síce pomerne priamočiare, ale existuje niekoľko variácií. V zadaní budeme vychádzať z pôvodných pravidiel s niekoľkými zjednodušeniami. V rámci riešenia najprv implementujete model hry (trieda `Piece` – reprezentuje figúrku, trieda `Board` – reprezentuje hracie pole, trieda `Person` – reprezentuje hráča, trieda `Game` – rieši hernú logiku). Následne implementujete niekoľko stratégií (`RandomPlayer`, `SafePlayer`, `MeanPlayer`, `EagerPlayer` – všetky podtriedy `Player`), a porovnáte ich výhernosť.

Triedy v riešení sú navzájom prepojené, poradie popisu predstavuje odporúčané poradie implementácie. Spomínané metódy a členské premenné sú minimálne požiadavky na triedy, ak potrebujete, môžete pridať ďalšie atribúty. Pri riešení musíte dodržať všetky princípy objektovo orientovaného programovania, najmä enkapsuláciu: nepristupujte k členským premenným priamo mimo triedy. V prípade porušenia enkapsulácie bude stiahnutých 20 percent bodov z hodnotenia.

## `Piece` – 0,5 bodov

Prvá trieda `Piece` reprezentuje figúrku, ktorá je definovaná tromi hodnotami:
- `color` (`string`) – farba figúrky, nastavená v konštruktore podľa farby hráča (platné hodnoty sú *black*, *yellow*, *green*, *red*);
- `active` (`boolean`) – vyjadruje, či figúrka stojí v stajni (`False`) alebo je na hracej ploche, prípadne v domčeku (`True`);
- `position` (`None`/`integer`) – vyjadruje pozíciu figúrky z jej pohľadu: stajňa je reprezentovaná hodnotou `None`, štartovacia pozícia je 0, hracie ploche je reprezentované číslami 0–39, pozície v domčeku majú číslo 40–43.

Do triedy potrebujete doplniť päť jednoduchých metód s nasledovnou funkcionalitou:
- `activate()` – metóda reprezentuje posunutie figúrky na štartovaciu pozíciu, upravte členské premenné `active` a `position` vhodným spôsobom.
- `throw_out()` – metóda reprezentuje vybitie figúrky, ktorá sa vráti do stajne, pričom aktualizujete premenné `active` a `position`.
- `is_done()` – metóda vracia `True` alebo `False` v závislosti od toho, či sa figúrka nachádza v domčeku.
- `move(steps)` – metóda vráti novú pozíciu figúrky po jej posunutí o `steps` počet polí na hracej ploche. Ustrážte limity platných pozícií (od 0–43). Zatiaľ neriešte, či je daná pozícia voľná alebo nie. Návratová hodnota teda vyjadruje číslo pozície, na ktorú by figúrka chcela prejsť. Ak figúrka zatiaľ nie je na hracej ploche, a padne 6, tak sa má presunúť na štartovaciu pozíciu (0). Ak figúrka stojí napríklad na pozícii 39 a má byť posunutá o 5 polí, tak ťah neplatí a figúrka ostane na pozícii 39.
- `move_to_place(position)` – metóda aktualizuje aktuálnu pozíciu figúrky podľa hodnoty parametra.

## `Board` – 2 body

Trieda `Board` reprezentuje hracie pole, ktoré je dané zoznamom `Board.board`, v ktorom každá hodnota je buď `None`, alebo objekt typu `Piece`, teda figúrka, ktorá na danej pozícii stojí. Hracie pole sa skladá zo 40 polí, domčeky ani stajne nie sú súčasťou poľa. Súbor okrem

toho definuje konštantu `STARTING_POSITIONS`, ktorá určuje polohu štartovacej pozície jednotlivých hráčov na hracom poli.

V triede potrebujete doplniť metódy:
- `normalize_position(position, player_color)` – pozícia z pohľadu figúrky (`Piece`) je vždy reprezentovaná číslom od 0 až 39, no keď sa pozrieme na celé pole, takáto reprezentácia nám nepostačuje. Metóda `normalize_position` prepočíta pozíciu z pohľadu figúrky (prvý parameter `position` typu `integer`) na pozíciu z pohľadu hracieho poľa na základe farby figúrky (druhý parameter `player_color` typu `string`). Ak figúrka stojí v stajni, pozícia bude naďalej `None`. V opačnom prípade prepočítajte polohu figúrky vzhľadom na štartovaciu pozíciu, a tak získate index prvku zoznamu `Board.board`, ktorý zodpovedá pozícii figúrky. Ako 0. pole sa berie štartovacie pole čiernych figúrok. Napríklad, ak žltá figúrka stojí na pozícii 4 z jej pohľadu, z pohľadu poľa to bude index 14, keďže žlté figúrky štartujú na indexe 10 (viď `STARTING_POSITIONS`). Metóda `normalize_position` musí vrátiť platný index pre zoznam `Board.board`, teda hodnotu medzi 0–39. Ak figúrka stojí v domčeku, jej pozíciu neprepočítajte, vráťte pôvodnú hodnotu (z intervalu 40–43).
- `can_move_there(player, piece, position)` – metóda vracia `True` alebo `False` v závislosti od toho, či hráč `player` môže presunúť svoju figúrku `piece` na pozíciu `position`, pričom `position` reprezentuje pozíciu z pohľadu figúrky (návratová hodnota `Piece.move`, a nie `normalize_position`). Metóda `can_move_there` teda vráti `True` pre platné ťahy, pričom ťah je platný, ak cieľová pozícia je prázdna, alebo na nej stojí figúrka protihráča. Do úvahy berte aj prípadné presunutie do domčeka – figúrka musí pristáť na poli, na ktorom ešte nie je figúrka rovnakej farby. V opačnom prípade metóda vráti `False`.
- `process_player_moves(player, targets)` – metóda aktualizuje hracie pole a figúrky na základe ťahu hráča `player`. Ťahy sú reprezentované v zozname `targets`, pričom každý prvok je n-tica s dvomi hodnotami, kde prvá hodnota je objekt typu `Piece` (teda figúrka, ktorú chce hráč posunúť) a `targets` vyjadruje pozíciu, na ktorú hráč chce figúrku presunúť (pozícia je zase reprezentovaná z pohľadu figúrky a nie z pohľadu hracej plochy). V metóde vyriešte presunutie figúrky a aktualizáciu poľa, pričom nezabudnite na potrebné úpravy:
    - vymazať z aktuálnej polohy posunutú figúrku,
    - ak na cieľovej pozícii stojí figúrka protihráča, tá má byť vyhodená a vrátená do stajne,
    - aktualizovať pozíciu posunutej figúrky,
    - vhodným spôsobom riešiť presunutie figúrky do domčeka.

## `Player` – 1 bod

Trieda `Player` reprezentuje hráča a definuje jeho všeobecnú funkcionalitu. Konkrétne stratégie výberu ďalšieho ťahu budú implementované v podtriedach. Hráč bude reprezentovaný nasledovnými hodnotami:
- `color` (`string`) – farba hráča (platné hodnoty sú *black*, *yellow*, *green*, *red*);
- `board` (`Board`) – hracie pole, na ktorom hráč hrá, referencia sa nastaví v konštruktore;
- `pieces` (`list` objektov `Piece`) – zoznam so štyrmi figúrkami hráča, ktoré sa vytvoria priamo v konštruktore; na začiatku hry sú všetky figúrky v stajni.

V triede potrebujete implementovať metódy:
- `is_done()` – vráti `True` v prípade, keď všetky hráčove figúrky sú v domčeku, teda hráč už nepotrebuje hodiť kockou. V opačnom prípade vráti `False`.

- `has_active()` – vráti `True` v prípade, ak má hráč aspoň jednu figúrku na hracom poli. V opačnom prípade vráti `False`.
- `get_piece_positions()` – vráti zoznam s `integer` hodnotami, pričom každý `integer` reprezentuje pozíciu jednotlivých figúrok (z pohľadu reprezentácie figúrky a nie poľa).
- `get_valid_moves(roll)` – vráti zoznam všetkých možných ťahov hráča na základe čísla, ktoré padlo po hode kockou a ktoré dostane metóda ako parameter `roll` (číslo typu `integer`). Návratová hodnota je teda zoznam, v ktorom každý prvok je n-tica s dvomi hodnotami: figúrka, ktorú hráč môže presunúť (objekt typu `Piece`) a počet krokov, o koľko ju posunie (`roll`). Jediná výnimka je, ak padne 6 a hráč má zatiaľ neaktívnu figúrku. V tomto prípade, ak je štartovacia pozícia voľná, hráč musí aktivovať prvú neaktívnu figúrku (vzhľadom na poradie v zozname `pieces`). Návratová hodnota bude zoznam s jedným prvkom: `[(prvá neaktívna figúrka, 0)]`.
- `prepare_targets(moves)` – metóda dostane ako parameter zoznam jednotlivých plánovaných ťahov (každý prvok bude n-tica s dvomi hodnotami ako v metóde `get_valid_moves`), a vráti zoznam ťahov, ktoré hráč neskôr vykoná, pričom pre každú figúrku bude mať jeden ťah. Prvky zoznamu budú n-tice s dvomi hodnotami: prvá hodnota je figúrka, ktorú chceme posunúť, a druhá hodnota bude číslo pozície, na ktorú ju chceme presunúť. Napríklad, ak ako parameter dostane zoznam `[(p1, 6), (p1, 6), (p1, 3)]`, kde `p1` je na začiatku neaktívna figúrka, tak návratová hodnota bude normalizovaný zoznam `[(p1, 9)]`, keďže prvá šestka sa použije na aktiváciu figúrky (postavíme ju na pozíciu 0), ďalším ťahom posunieme figúrku o 6 polí, a posledným ťahom o ďalšie 3. V zozname teda budeme mať vyjadrené, ktorú figúrku na ktorú pozíciu chceme posunúť. Keby sme mali zoznam `[(p2, 6), (p2, 2)]`, pričom na začiatku figúrka `p2` stojí na pozícii 14, tak návratová hodnota bude `[(p2, 22)]`.
- `get_move(roll)` – metóda vyjadruje rozhodnutie hráča, ktorý ťah vykoná. Metódu v tejto triede neimplementujte, bude závisieť od konkrétnej stratégie, bude teda implementovaná v podtriedach.

## `Game` – 2 body + 0,5 bodov

Trieda `Game` rieši celkovú hernú logiku a nasimuluje jednu hru s rôznymi typmi hráčov. `Game` bude daná hodnotami:
- `board` (`Board`) – reprezentuje hracie pole;
- `players` (`list` objektov typu `Player` a podtried) – zoznam hráčov, ktorí sa vygenerujú na základe parametra `player_types`, ktorý je zoznam so štyrmi hodnotami určujúcimi typ hráčov (konkrétne typy implementujete neskôr). Zoznam sa úplne pripraví v konštruktore, nemusíte ho upravovať.

Do triedy potrebujete doplniť dve metódy:
- `get_player_rolls(player)` – vráti zoznam hodov (zoznam `integer`ov) pre hráča `player` (objekt typu `Player` alebo podtriedy). Dĺžka zoznamu bude daná stavom hráča. Ak hráč má aktívnu figúrku na hracej ploche, tak hodí iba raz. Ak padne 6, môže hodiť ešte raz (ak zase padne 6, hod zopakuje, atď.). Ak hráč aktívnu figúrku na hracej ploche nemá, hodí maximálne trikrát. Ak skôr padne 6, postupuje podľa všeobecných pravidiel (hod zopakuje, kým padá 6).
- `run_game()` – metóda nasimuluje jednu hru, a vráti dve hodnoty: počet kôl potrebných na ukončenie hry (typu `integer`) a poradie hráčov (zoznam objektov typu `Player` a podtried). Hra prebieha dovtedy, kým nemajú všetci hráči všetky svoje figúrky v domčeku. Hráči hodia

kockou v každom kole striedavo podľa zoznamu `players` (pre rýchlejšie vykonávanie odporúčame brať do úvahy iba ešte hrajúcich hráčov), pričom najprv získajte hody hráča, potom každý hod spracujte po jednom: získajte ťahy hráča a aktualizujte hracie pole na základe ťahu. V našej verzii hry figúrku protihráča môžete vyhodiť aj v prípade, ak svojou figúrkou pokračujete ďalej. Ak hráč úspešne ukončil hru, pridajte ho do zoznamu `order`, teda do finálneho poradia.

Okrem triedy `Game` súbor obsahuje funkciu `find_best_strategy(player_types, runs)`, ktorá nájde stratégiu, ktorá vyhrá najčastejšie. Funkcia má dva parametre, prvý z nich je zoznam s typmi hráčov (podobne ako pri konštruktore `Game`), druhý parameter `runs` udáva počet nasimulovaných hier, na základe ktorých sa nájde najviac výherná stratégia. Funkcia má dve návratové hodnoty: prvá hodnota je typ hráča, ktorý vyhral najčastejšie; druhá hodnota je `dictionary` s prehľadom počtu výhier rôznych stratégií. `Dictionary` bude obsahovať štyri kľúče, ktoré budú totožné s typmi zo zoznamu `player_types`, a hodnoty uložené pod kľúčmi budú celé čísla vyjadrujúce počet hier, ktoré vyhral daný typ hráča (súčet teda musí rovnať `runs`). Pre férové porovnanie stratégií poradie hráčov premiešajte náhodne pred každou simulovanou hrou.

## Stratégie – 4 × 1 bod

Ostáva už len implementácia podtried `Player`, ktoré definujú rôzne stratégie pri hraní hry človeče. V každej triede implementujete iba metódu `get_move(roll)`, ktorá vráti vybraný ťah hráča na základe padnutého čísla `roll`. Návratová hodnota je teda zoznam s jednou n-ticou, kde n-tica obsahuje figúrku, ktorú chce hráč posunúť (objekt typu `Piece`) a číslo pozície, na ktoré ju chce posunúť (`integer`, vyjadruje pozíciu z pohľadu figúrky a nie z pohľadu poľa).

Konkrétne implementujte nasledovné správania:

- `RandomPlayer` – z dostupných ťahov vyberie vždy náhodne.
- `SafePlayer` – snaží sa dostať do domčeka čo najskôr jednu figúrku. Teda vždy vyberie ťah, ktorý posunie figúrku, ktorá je najbližšie k dokončeniu kruhu. Aj preňho však platí pravidlo, že ak štartovaciu pozíciu má voľnú a padne 6, musí na štart položiť figúrku (ak ešte má neaktívnu).
- `MeanPlayer` – ak má možnosť vybiť figúrku protihráča, urobí tak, v opačnom prípade vyberie náhodný ťah.
- `EagerPlayer` – chce dostať čo najviac figúrok na hracie pole, a práve preto ak má figúrku na štarte, tak ju posunie (ak je to možné). Takisto sa snaží vybiť figúrky protihráča, takže ak sa mu poskytne príležitosť, tak figúrku vybije. Ak už má všetky svoje figúrky na hracom poli a nikoho nemôže vybiť, vyberie ťah náhodne.

**Do tried môžete pridať ďalšie vnútorné premenné, musíte tam však nechať tie, ktoré sú dané zadaním** (inak testy neprejdú a dostanete 0 bodov za danú triedu). Dodržujte princípy OOP a nepristupujte priamo k členským premenným mimo triedy.