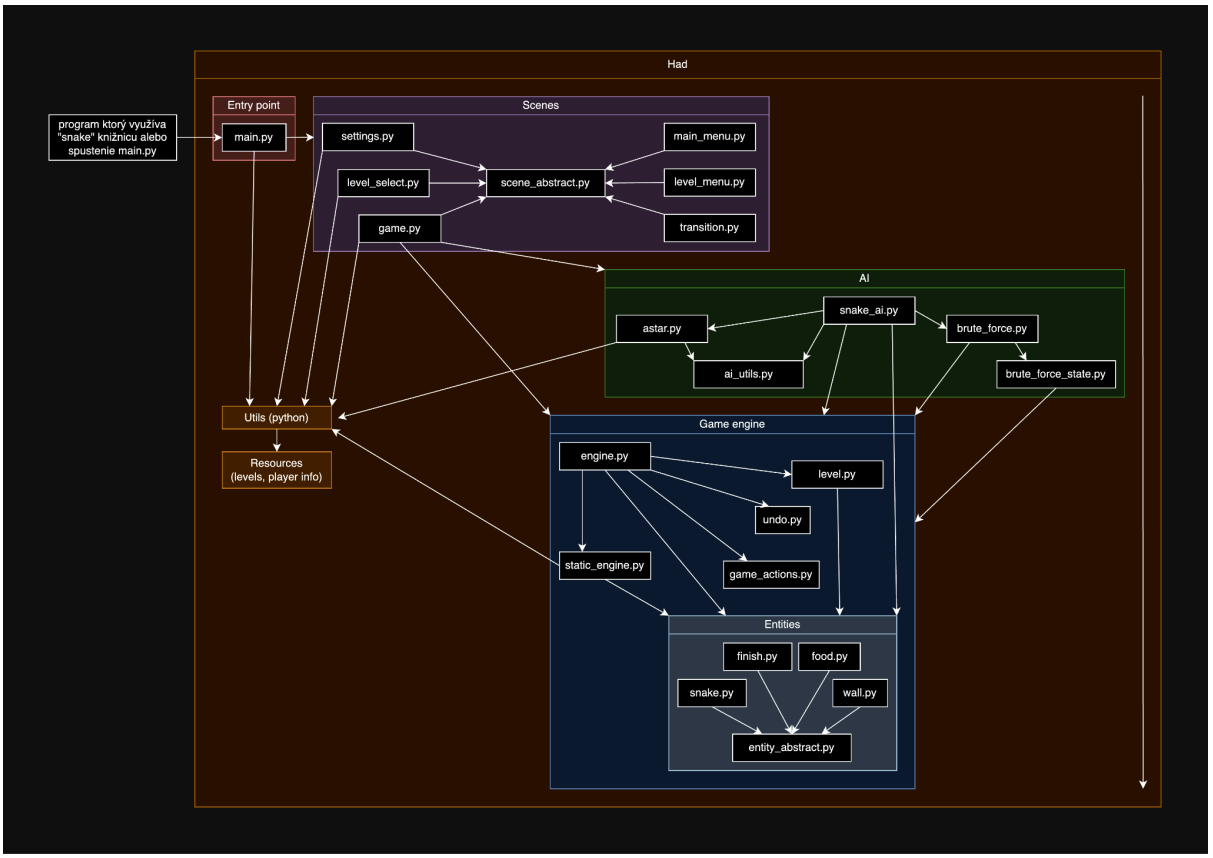
# Technická dokumentácia

Had

## Prehľad

Aplikácia had sa skladá z viacerých modulov kde každý zabezpečuje časť funkcionality celej aplikácie. Takto vyzerá jej štruktúra v grafe:




A → B na tomto grafe znamená že A používa/importuje B

Toto robia jednotlivé časti:

- *main.py* je hlavný riadiaci bod aplikácie, stará sa o to aby aplikácia mala kde ukazovať výstup, aby sa správne menila veľkosť okna aplikácie, aby sa správne spracoval vstup používateľa, aby sa na výstupe ukazovalo všetko potrebné a spravuje komunikáciu medzi jednotlivými scénami.
- *scény* sú jednotlivé režimy ktoré práve aplikácia vykonáva medzi ktorými sa dá ľahko prepínať ako napríklad *menu*, *hra*, *nastavenia.* Každá scéna riadi niečo iné a má svoje vlastné spracovanie a grafický výstup. Najdôležitejšia je scéna `Game` ktorá umožňuje používateľovi hrať hada.
- *umelá "inteligencia"* riadi *autoplay* režim aplikácie - teda snaží sa nájsť cestu pre hada do cieľa pomocou rôznych algoritmov bez zásahu uživateľa.
- *game engine* (to neviem ako by som preložil) riadi beh hry, najdôležitejší je tu objekt `Engine` ktorý dostáva ako vstup pohyb hada (nerieši či od človeka alebo od algoritmu) a spracuje ho v rámci pravidiel hry. Tiež dôležitý je tu objekt

`StaticEngine` ktorý pri spustení levelu predspracuje stav hry čo umožňuje jednoduchý a rýchly výpočet napríklad kolízií
- *entity* sú najmenšie stavebné bloky interného riadenia hry - sú to jednotlivé prvky hry ako napríklad *had*, *stena*, *jedlo*, tu sa nič zaujímavé z technickej stránky nedeje.
- *utils* rieši doplnkovú funkcionalitu ktorá mi nikde inde nesedela. Napríklad načítavanie levelu zo súboru, načítavanie a ukladanie dát medzi spusteniami a rozdeľovanie objektov na komponenty podľa spojitosti
- *resources v sebe nemá žiadny kód ale obsahuje definície jednotlivých levelov a ukladajú sa tam informácie o používateľovi aby bola aplikácia v rovnakom stave ako ju používateľ zanechal*

dopodrobna budú jednotlivé časti vysvetlené v opačnom poradí - v takom ako program dané objekty definuje aby sa na nich dalo potom stavať.

# Varovanie

V tomto súbore budem písať o veciach ako elektrika, ktorá má vlastné pravidlá v rámci hada alebo napríklad entity ktoré môžu hada zraniť a v tomto bode to do hada ešte vôbec nie je implementovaná takže to netreba moc riešiť.

# Resources

Sú tu dva druhy súborov
1) .hadik - V tomto súbore sú uložené všetky dáta ktoré je potrebné na spustenie levelu takže jeho šírka, výška, camera offset, začiatočná pozícia hada a všetky entity
2) .save - V tomto súbore sú uložené zvolené nastavenia používateľa medzi spusteniami aplikácie čo sú fullscreen a autoplay možnosti. Tiež je tu uložené ktoré levely už sú odomknuté lebo na začiatku je odomknutý len prvý level.

# Utils

## player_data.py

Definuje triedu `PlayerData` ktorá má v sebe informácie o momentálnom stave fullscreenu, autoplay režimu a odomknutých leveloch, jej prvá metóda `load()` načíta dáta zo súboru player_data.save ktorý sa nachádza v snake/resources. Jej druhá metóda `save()` do tohoto súboru zase všetky aktuálne dáta uloží.

## load_level.py

Implementuje jedinú funkciu:

```
load_level(level_number) -> tuple[Level, int, int]
```

Táto funkcia prečíta súbor `f"{level_number}.hadik"` ktorý sa nachádza v snake/resources a vráti inicializovaný level ktorý sa dá použiť na hranie hada. Tie zvyšné

dva inty čo vracia sú camera offset x, y pretože tie nesúvisia s hraním daného levelu, len s jeho ukazovaním na obrazovku. Súbory *.hadik majú svoju špecifickú štruktúru ktorá musí byť dodržaná a momentálne nie je žiadny rozumný spôsob ako si spraviť svoj level.

## group.py

Implementuje dve funkcie pre rozdeľovanie mnoho vecí na menšie skupiny podľa toho ktoré sú spojené.

```
get_connected_blocks(blocks: list[tuple[int, int]]) -> list[list[tuple[int, int]]]:
```

je klasický algoritmus na rozdeľovanie 2D grafu na komponenty podľa spojitosti. Nič zvláštne. Používa ho umelá “inteligencia” na hľadanie cesty pre hada, napríklad iný algoritmus vráti 20 možných pozícií na ktoré teoreticky vie had z momentálnej pozície prejsť ale keď sú rozdelené (povedzme stenou) tak had vie ísť len na pozície z tej skupiny kde sa sám nachádza.

```
get_connected_conductive_groups(entities: list[StaticEntity]) -> list[list[StaticEntity]]:
```

Robí zhruba to isté ale využíva vlastnú logiku v hadovi na rozdelenie statických objektov v leveli na skupiny ktoré sú navzájom vodivé, to znamená že sú dosť blízko seba na to aby zdieľali elektrický náboj. Túto funkciu používa statický engine na predspracovanie levelu na elektrické grupy čo bude asi vysvetlené neskôr.

# Entity

## entity_abstract.py

Definuje materskú triedu `Entity` z ktorej dedia všetky ostatné entity, definuje všeobecné správanie entity. Všetky jej metódy sú abstraktné aby sa zabezpečilo že z nej zdedené entity budú implementovať všetko čo by entita mala vedieť. Má tieto metódy:

```
__init__(self, conductive: bool, charge: bool)
```

Parametre určujú či táto entita je vodivá a ak hej, či je hneď nabitá elektrikou.

```
draw(self, canvas: Canvas, paddingx: int, paddingy: int, block_size: int) -> None
```

Definuje ako bude entita vyzerať v hre, parametre *paddingx, paddingy* a *block_size* slúžia na konverziu z herných súradníc (napríklad (2, 3)) na súradnice plátna canvas.

```
get_collision_coords(self) -> list[tuple[int, int]]
```

Definuje či a ako je entita pevná. Nedá sa prejsť cez súradnice ktoré vráti táto funkcia.

```
get_hurt_coords(self) -> list[tuple[int, int]]
```

Definuje či a ako je entita nebezpečná. Prejdenie cez súradnice ktoré vráti táto funkcia odreže článok hada ktorý cez ne prešiel.

```
get_electricity_coords(self) -> list[tuple[int, int]]
```

Definuje či a ako entita šíri elektriku. Keď je entita nabitá elektrinou aj súradnice ktoré vráti táto funkcie sú nabité elektrinou.

```
get_interact_coords(self) -> list[tuple[int, int]]
```

Definuje či a ako sa dá interagovať s touto entitou. Keď je had na súradniciach ktoré vráti táto funkcia, niečo sa stane.

```
get_interact_type(self) -> "Entity.InteractType"
```

Definuje aká interakcia sa stane keď had interaguje s touto entitou. Možnosti sú enum typu `Entity.InteractType` ktorý má hodnoty *NONE*, *FINISH*, *FOOD*, *CHECKPOINT.*

Okrem toho tento súbor definuje aj dva typy entít ktoré dedia z `Entity`:

1) `StaticEntity` - entita ktorá má obdĺžnikový tvar a nemení pozíciu
2) `DynamicEntity` - entita ktorá môže mať ľubovoľný tvar a môže sa aj hýbať a môže na ne pôsobiť gravitácia

## snake.py

Implementuje triedu `Snake` ktorá dedí z `DynamicEntity`. Je to entita ktorá má dynamický tvar, vedie elektrinu a pôsobí na ňu gravitácia. Je tvorená z fronty jednotlivých blokov aby bolo jednoduché pohybovať hadom tak ako sa had v hre pohybuje (chvost v predu sa každým pohybom popne a appendne sa nová hlava). Had je modrý keď ním neprechádza elektrika a zlatý keď ním prechádza.

## wall.py

Implementuje triedu `Wall` ktorá dedí zo `StaticEntity`. Je to stena ktorá vedie elektrinu a had cez ňu neprejde.

## food.py

Implementuje triedu `Food` ktorá dedí zo `StaticEntity`. Je to jedlo ktoré nevedie elektrinu a predĺži hada o 1 článok.

## finish.py

Implementuje triedu Finish ktorá dedí zo StaticEntity. Je to cieľ, ktorého dosiahnutie úspešne ukončuje level.

# Game engine

## undo.py

Implementuje triedy EatenFood, EntityPosition a Undo. Undo je hlavná trieda ktorá na svoj beh využíva zvyšné dve triedy.

Trieda EatenFood si pamätá event ktorý sa stal (eventuálne by eventov malo byť viac ako len jedenie jedla) a kde sa stal, teda (x, y) súradnice kde jedlo bolo zjedené.

Trieda EntityPosition si pamätá pozíciu nejakej dynamickej entity. Dynamické entity okrem samotného hada môžu mať pestré tvary ale keďže sa ich tvar počas behu levelu nemení stačí si pamätať ich (x, y) súradnice. Tiež si ukladá referenciu na samotnú entitu aby mohla zmeniť jej súradnice v prípade že treba ísť naspäť v čase.

Trieda Undo si pamätá jeden frame vecí ktoré sa v hadovi stali, normálne je vracanie sa naspäť v čase možné len v debug móde ale umelá “inteligencia” ho využíva na brute force algoritmus takže je to potrebné. Táto trieda má 3 hodnoty:

```
snake: deque[tuple[int, int]]
```

Keďže had vie meniť svoj tvar pamätá si undo pozíciu všetkých jeho článkov v danom frame.

```
dynamic_entities: list[EntityPosition],
```

Pamätá si pozíciu všetkých dynamických entít v danom frame.

```
events: list[EatenFood]
```

Pamätá si eventy ktoré sa v danom frame stali, zatiaľ len jedenie jedla.

# game_actions.py

Implementuje jednoduchý enum `Action` všetkých inputov ktoré game engine pozná. Každý frame game engine dostane jeden z týchto inputov a na základe toho beží celá hra. Táto trieda predstavuje interface cez ktorý aplikácia hovorí game enginu čo má robiť.
Má tieto hodnoty:
DO_NOTHING - Had sa nehýbe, jedine že by padal kvôli gravitácii
MOVE_LEFT - Skús pohnúť hadom o jedno políčko doľava
MOVE_RIGHT - Skús pohnúť hadom o jedno políčko doľava
MOVE_UP - Skús pohnúť hadom o jedno políčko dohora
MOVE_DOWN - Skús pohnúť hadom o jedno políčko dolu
STOP_MOVEMENT - Zastav automatické procesy ako padanie. Povolené len v debug režime.
UNDO_MOVEMENT - Vráť sa o jeden frame naspäť. Povolené len pre umelú “inteligenciu” alebo v debug režime.

# level.py

Implementuje jednoduchú triedu `Level` ktorá uchováva všetky informácie o momentálnom stave levelu. Má hodnoty pre výšku, šírku levelu, pozíciu a veľkosť hada (fronta jeho článkov) a samostatné hodnoty pre statické a dynamické entity. Statické entity sa spracujú na začiatku levelu a ďalej sa len vykresľujú, dynamické entity sa spracúvajú každý frame.

# static_engine.py

Som unavený, dokončím to zajtra

engine.py

# Umelá “inteligencia”

Ako to funguje?

brute_force_state.py

brute_force.py

ai_utils.py

astar.py

snake_ai.py

## Scény

scene_abstract.py

game.py

main_menu.py

level_select.py

level_menu.py

settings.py

transition.py

## Aplikácia (main.py)