SNAKE

Anotace, úvod do problému, popis programu

Maturitní práce z Informatiky
Martin Bláha R8.A
2019
Vedoucí práce: Karel Jílek

Obsah:

Zadání:

Zadáním bylo vytvořit emulátor tradiční arkádové hry Snake pro desktop. Hra má uživateli nabízet různé úrovně obtížnosti pomocí překážek či rychlejšího pohybu hada.

Software:

Program jsem vytvořil ve verzi Pythonu 3.6.8 s použitím grafické knihovny Qt, resp. PyQt5. Pro vývoj jsem použil vývojový nástroj JetBrains PyCharm Community Edition 2018.3.4.

Pravidla hry:

Had, tvořený několika články, se pohybuje po předem vymezeném hracím poli, které se skládá ze čtvercových políček stejné velikosti (tzv. mapa), přičemž všechny objekty (články hada, jídlo, překážky) jsou tvořeny těmito unifikovanými políčky. Ovládání pohybu je zcela prosté - pomocí šipek či kláves W,A,S,D hráč určuje směr pohybu hada, tzn. nahoru, doleva, dolů nebo doprava. Náhodně v hracím poli je vždy vygenerováno jídlo (červený čtvereček o velikosti jednoho políčka). Cílem hráče je nasměrovat hada k jídlu a při tom nenarazit do stěny hracího pole, do překážek ani do ostatních článků hada (v takovém případě se hra ukončí). Pokud had "sní" jídlo, tzn. hlava hada přejede přes políčko s jídlem, zvýší se skóre a úroveň, přibude mu na konec jeden článek navíc a zvýší se rychlost pohybu, což činí hru o něco náročnější. Tímto způsobem hráč postupuje do dalších úrovní a odemyká nové mapy, které poskytují např. větší hrací plochu nebo jiné rozložení překážek (každých 5 úrovní se odemkne nová mapa - první mapa je bez překážek "na rozehřátí", poslední mapa obsahuje málo překážek, ale je nekonečná, tzn. že se stále zvyšuje rychlost a zvětšuje délka hada a hráč hraje na této mapě dokud to jde). Pokud had narazí do stěny, do překážky nebo sám do sebe, hra se ukončí. Hráč uvidí dosažené skóre v poslední hře a také nejvyšší skóre ze všech předchozích her. Poté se může rozhodnout, zda hrát znovu nebo program opustit (po opuštění programu se maximální skóre vynuluje).

Popis programu pro uživatele:

Hrací pole je uloženo v paměti jako soubor jednotlivých políček, z nichž každé obsahuje číslo - informaci o tom, který objekt se na daném políčku právě nachází, tedy jestli na něm je had, jídlo, překážka nebo je prázdné. Program při každém pohybu toto pole celé vykreslí na obrazovku.
Ovládání je zajištěno ukládáním poslední stisknuté klávesy do proměnné, podle které se určí směr pohybu hada.
Při každém pohybu hada program zkontroluje, zdali had nenarazil do překážky, do kraje herního pole nebo sám do sebe - pokud ano, tak se hra ukončí, pokud ne, tak se provede pohyb. Dále se zkontroluje, jestli had náhodou nesnědl jídlo (tj. nachází se na stejných souřadnicích jako jídlo) a pokud ano, tak provede příslušné akce - zvýšení skóre a úrovně, prodloužení hada a zvýšení rychlosti).
Každých 5 úrovní program načte novou mapu a chvilku (1,5 s) počká, aby se hráč mohl "rozkoukat" - hráč tak nemusí rychle měnit směr pohybu, neboť počáteční pozice je vždy nastavena tak, aby hráč ihned nenarazil do překážky/zdi a měl tak dostatek času na reakci.
Při ukončení hry se zobrazí obrazovka s nápisem "Konec hry!" a hráč uvidí své současné skóre a také nejvyšší dosažené skóre. Zároveň má hráč možnost hrát znovu (klávesa Y - Yes) nebo již nehrát (klávesa N - No).

Popis programu pro programátory:

Program má dvě základní části - grafickou a logickou. Grafická část zajišťuje uživatelské rozhraní a logická zajišťuje běh hry.

Grafická část v každém herním cyklu projde všechna políčka hracího pole a každé políčko v závislosti na hodnotě, která je v daném políčku uložená, vykreslí příslušnou barvou (pomocí QPainteru), případně jej nechá prázdné - nevybarvené.

- 0 = prázdné políčko (bezbarvé)
- -1 = jídlo (červené)
- -2 = překážka (černá)
- 1 = hlava hada (modrá)
- >1 = tělo/články hada (světle modrá)

⇒ v případě ukončení hry zobrazí obrazovku s nápisem "Konec hry", s dosaženým skóre, nejvyšším dosaženým skóre a možnostmi "Hrát znovu? Yes/No" (viz screenshoty). Použil jsem mezinárodní Yes/No, protože Jo/Ne vypadá divně a nepoužívá se (A jako Ano jsem nepoužil, protože klávesa A zároveň ovládá směr pohybu a kdyby náhodou hráč při úhybném manévru na poslední chvíli použil právě klávesu A, avšak příliš pozdě na to, aby odvrátil srážku, nechtěně by tím restartoval hru, což není žádoucí).

Logická část se stará o vše ostatní:

1) Herní pole - je tvořeno polem polí, resp seznamem seznamů
   - ➢ hlavní seznam reprezentuje celé hrací pole
   - ➢ seznamy uvnitř reprezentují řádky hracího pole
   - ➢ položky uvnitř těchto seznamů reprezentují jednotlivá políčka

- pro přístup na určité políčko pomocí příkazu *pole[][]* musíme do prvních závorek zadat Y-souřadnici a do druhých závorek X-souřadnici, protože nejdříve vybíráme řádek a pak až sloupec (konkrétní políčko)

2) Jídlo - generování jídla na náhodných souřadnicích zajišťuje metoda *Jidlo()*, která vygeneruje náhodné souřadnice a poté ověří, zda se na

daném políčku již nenachází had nebo překážka. Pokud vygenerované souřadnice jsou obsazené, opakuje proces, dokud nenajde takové souřadnice, které nejsou obsazené - poté do nich (do odpovídající položky v seznamu) uloží hodnotu -1.

3) Ovládání - hada lze ovládat pomocí šipek nebo kláves W,A,S,D
   - ➢ zpracování vstupu z klávesnice zajišťuje metoda *keyPressEvent()*
   - ➢ při stisknutí šipky nebo klávesy se do proměnné *next_state* uloží příslušný směr hada a při pohybu hada se tento směr uloží do proměnné *current_state* a provede se pohyb příslušným směrem
   - ➢ v případě potřeby lze program kdykoliv ukončit stisknutím klávesy Escape

4) Pohyb hada - ten zajišťuje metoda *HniSe()*
   - ➢ tato metoda je napojena na výstup *QTimeru*, jehož interval určuje délku jednoho herního cyklu a tím i rychlost pohybu hada
   - ➢ na začátku se proměnná *next_state* (v níž je uložen poslední vstup z klávesnice) uloží do proměnné *current_state*, podle které se bude vykonávat pohyb v současném herním cyklu
   - ➢ poté se posune tělo - ke každému článku hada, reprezentovaném číslem v seznamu políček, se přičte 1 a poté se smaže poslední článek (číslo posledního článku je delší než délka hada)
   - ➢ poté se zkontroluje, jestli při pohybu hlavy had do něčeho nenarazí:
     - ■ pokud ano, hra se ukončí
     - ■ pokud ne, posune se hlava hada směrem, který je uložen v proměnné *current_state* a zapíše se do příslušné položky v *poli* jako 1
   - ➢ poté se zkontroluje, jestli had nesnědl jídlo
     - ■ pokud ano, tak se zvýší úroveň, skóre, délka hada a rychlost hada (resp. sníží se čas trvání herního cyklu) a vygeneruje se nové jídlo. Zároveň se každých 5 úrovní odemkne nová mapa a zvýší se bonus, který se přičítá ke skóre (to pak narůstá rychleji)
     - ■ pokud se odemkne nová mapa, program se na chvilku zastaví, aby umožnil hráči zareagovat na změnu, a poté

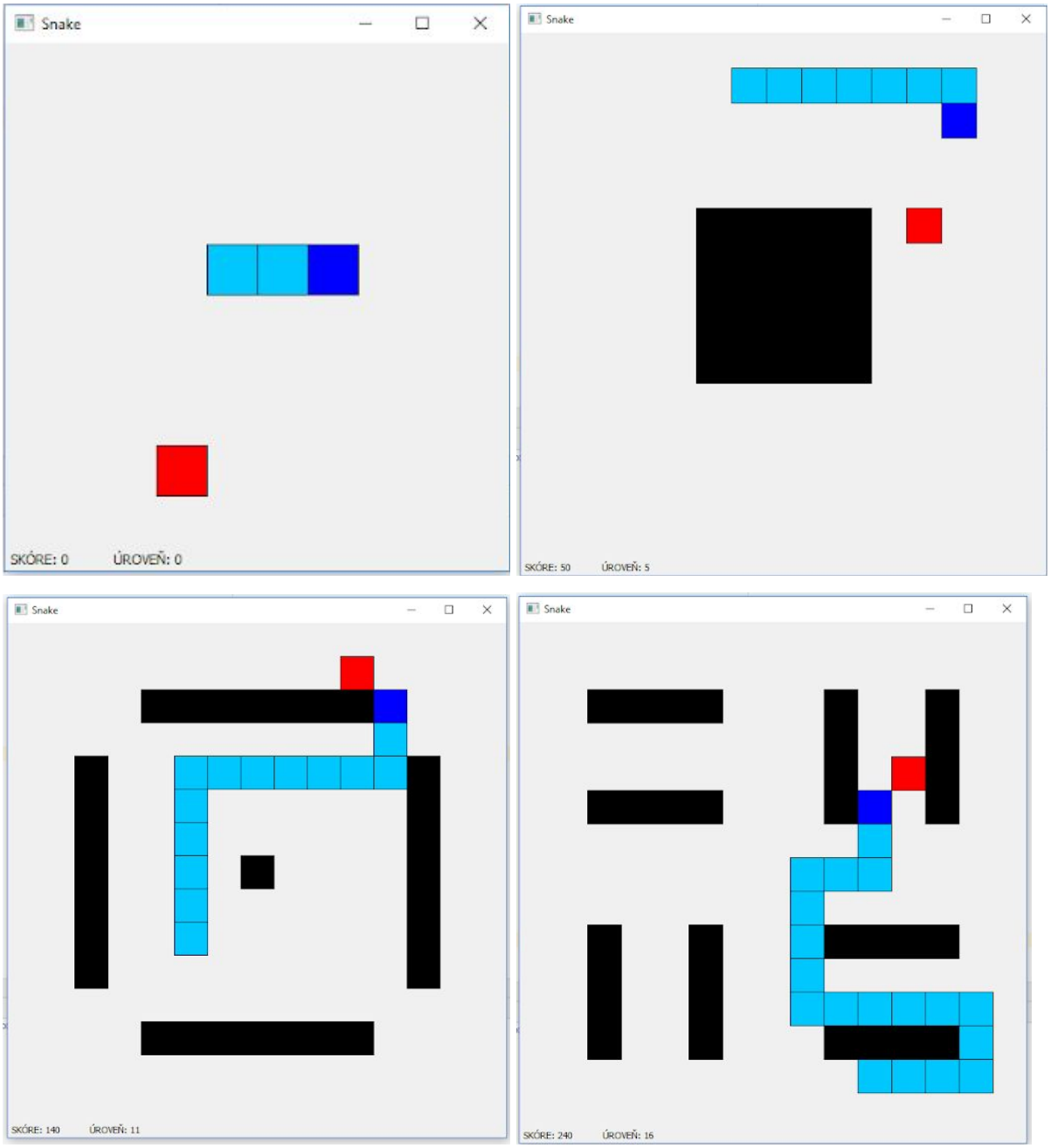
normálně pokračuje dál. Toto je zajištěno díky funkci *time.sleep()* a proměnné *pozastaveni* jejíž hodnota je defaultně 0, ale pokud se objeví nová mapa, uloží se do ní hodnota 1, což aktivuje podmínku, jejíž tělo obsahuje funkci *time.sleep()* - hned po provedení funkce *time.sleep()* se hodnota *pozastaveni* nastaví zpět na nulu, aby nedocházelo k zastavování v každém herním cyklu

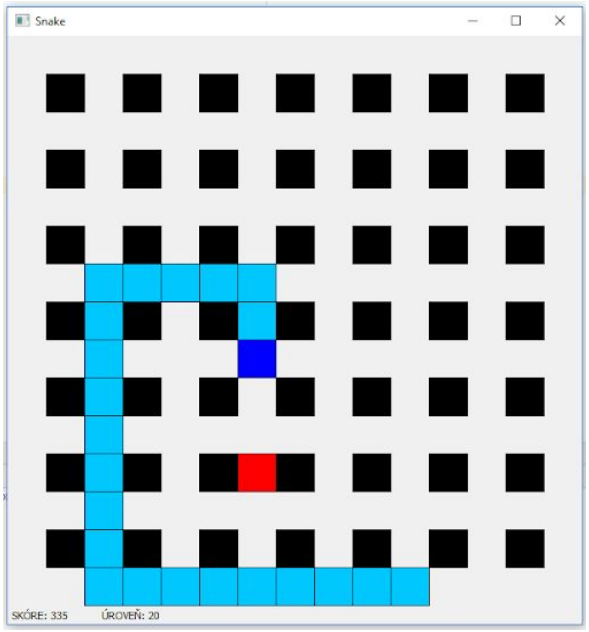
5) Vycentrování okna - to zajišťuje metoda *center_the_screen()*, která je zavolána vždy, když se mění velikost okna
   - ➢ zjistí velikost obrazovky počítače - šířku a výšku, od těch odečte šířku a výšku okna a tuto hodnotu vydělí dvěma. Na tyto souřadnice poté umístí okno.

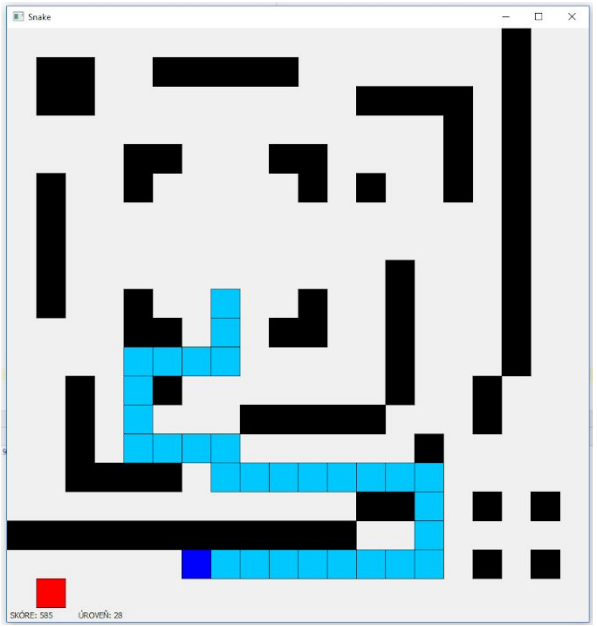
6) Mapy - ty jsou generovány pomocí funkce *mapa_N()* (kde N je číslo úrovně, na které se má tato funkce zavolat)
   - ➢ příslušná *mapa* je zavolána na každé páté úrovni

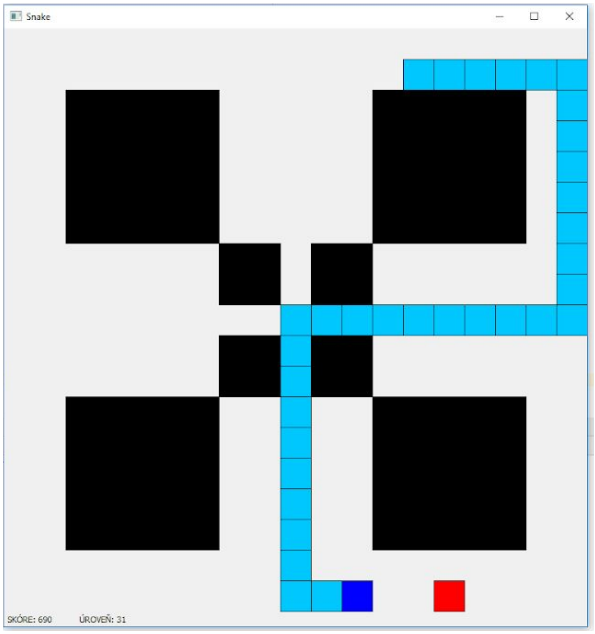
Screenshoty:

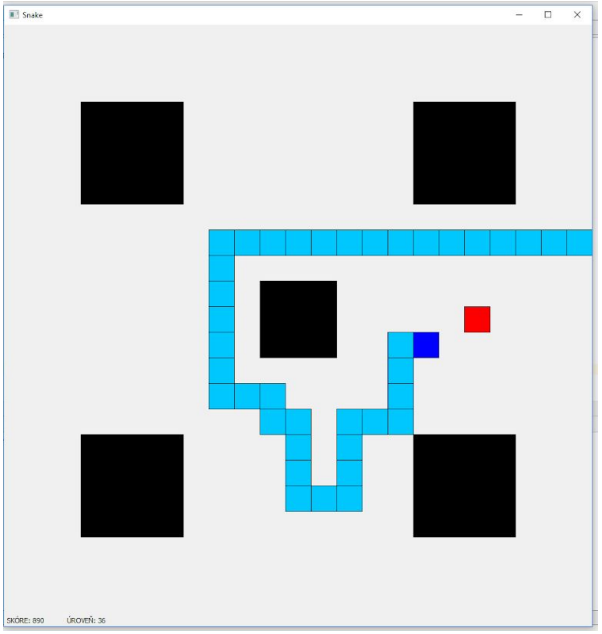
Screenshoty jednotlivých map:



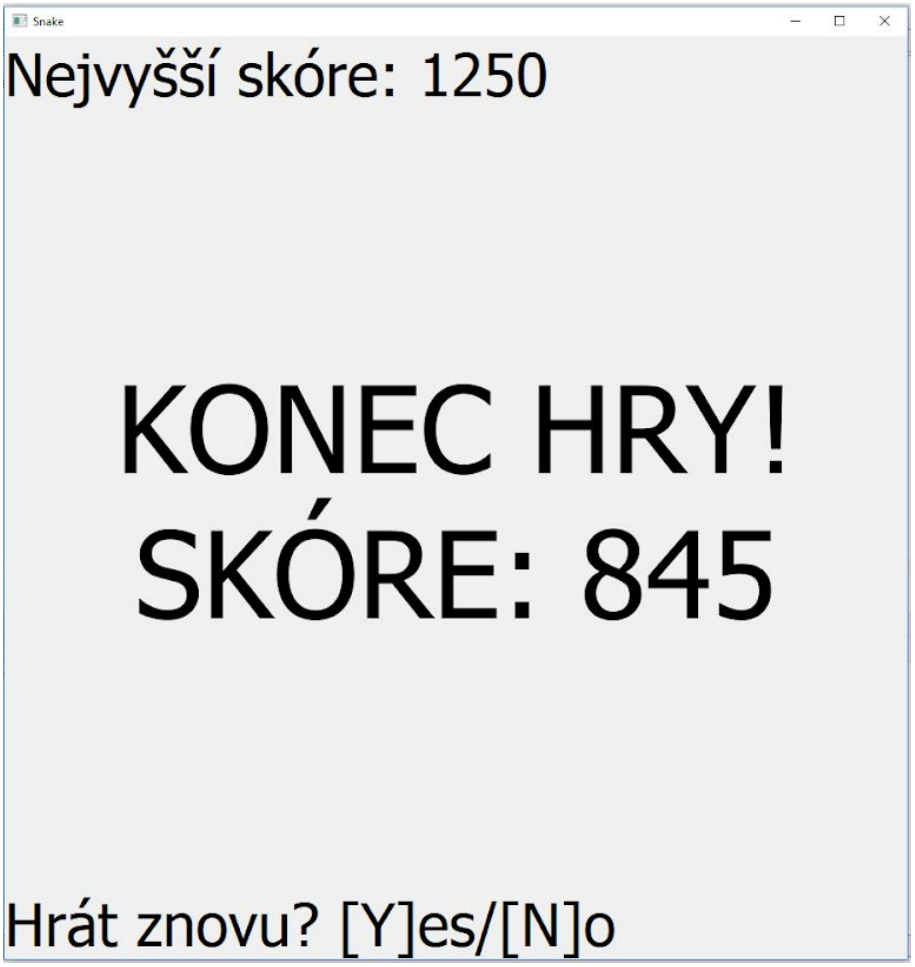

Snake
SKÓRE: 335
ÚROVEŇ: 20


Snake
SKÓRE: 585
ÚROVEŇ: 28


Snake
SKÓRE: 690
ÚROVEŇ: 31


Snake
SKÓRE: 890
ÚROVEŇ: 36

Screenshot obrazovky při prohře:



Závěr:

Pro generování map by bylo optimální použít předem vyplněné pole (již s překážkami) místo *for cyklů*, neboť ty jsou pomalejší.
Do programu by šlo přidat inteligentní generování náhodných map (tj. map s náhodně rozmístěnými překážkami, avšak tak, aby netvořily např. slepou uličku nebo nedostupnou uzavřenou oblast). Další užitečnou funkcí by byla možnost pozastavení a opětovného spuštění hry.