Anden porteføljeopgave: Snake

Du har besluttet dig for at lave din egen version af spillet Snake. For at lære at håndtere kompleksitet, laver du med vilje ikke spillet i en game engine, men istedet i Python med OpenCV. (se evt. et snake spil her: https://playsnake.org/).

Snake-verdenen består af et grid, slange-hovedet befinder sig i en celle i griddet, og efter en bestemt tid, rykker slangehovedet sig i den retning, spilleren sidst har markeret med tasterne 'wasd'. I banen er der også et æble, og når slangens hoved rammer æblet, spiser slangen æblet, slangens krop gror med én celle i længden, og slangens krop følger efter hovedet. Når æblet bliver spist, dukker der et nyt æble op i en tilfældig celle i griddet. På skærmen vises også en score, som er antallet af æbler, slangen har spist. Hvis slangen rammer siderne i griddet eller sin egen krop, dør slangen og spillet stopper.

Du har skrevet et kort script, som gør det ovenstående (*snake_script.py*), men spillet er lidt kedeligt i længden. Du får den idé, at man skal kunne vinde, hvis man får spist nok æbler. Du får også idéen, at der skal være flere levels med forskellige features, og du har følgende idéer til features:

1. Nogle af griddets celler er vægge, som slangen dør af at ramme, og hvor der ikke kan spawne æbler.
2. Når slangen rammer kanten, bliver den transporteret om på den anden side istedet for at dø.
3. Hastigheden bliver hurtigere jo flere æbler, der er spist.
4. Æbler forsvinder efter en tid og dukker op et tilfældigt andet sted.
5. Udover et almindeligt æble, er der også et æble med orm i, som slangen dør af at spise.
6. Ved at trykke på "mellemrum" skifter visualiseringen af slangen og æbler mellem cirkler (som i scriptet) og firkanter.

Opgave

Implementér "levels", mindst to af idéerne ovenover og mindst én egen idé ved brug af et passende software design med hensyn til at opnå pålidelig og vedligeholdbar kode.

1. Når man gennemfører et level, skal det næste level starte automatisk.

2. Det første level skal være ligesom i scriptet.
3. De resterende levels skal bestå af kombinationer af dine features.
4. Koden skal løbende refactors med hensyn til den kompleksitet, der kommer af de features, du vælger.
5. Lav en kort rapport, der indenholder:
    - Hvilke idéer til features, du har valgt.
    - Dokumentér lidt at processen. Prøvede du noget, som viste sig ikke at virke? Hvad virkede?
    - Giv nogle eksempler på, hvad du endte ud med i dit software design:
        - Ét eller flere UML diagrammer af relevante interfaces, klasser og deres relationer (Du behøver ikke at have alt med eller angive metoder og attributter).
        - Mindst ét eksempel på et designmønster, du har brugt, og overvejelserne bag det.
6. Aflever rapport ("rapport.pdf") og kode ("kode.zip").

Hjælp

- Det kan være en hjælp at bruge versionsstyring og test-drevet udvikling.
- Læs det materiale, der er lagt op til lektion 7 (lektion7.zip).
- Hvis strategimønstret stadig er "fluffy", så læs kapitel 7 i FRS, som en god del af lektion 7 er bygget op om.
- I kan bruge PlantUML til UML diagrammet.
- Jeg har lagt en delvis refactor op (mappen *snake_oop*), som måske kan hjælpe jer hen imod at have levels. I behøver ikke bruge det. Det er kun ment som en hjælp til at komme i gang - og bemærk, at designet ikke nødvendigvis passer til de features, i skal implementere.

Kommentar

Jeg er kommet frem til den her opgave, fordi

- Det er svært at finde en rigtig robot-case, som passer i kompleksitet til en portefølje (Det bliver meget hurtigt meget mere komplekst).
- Den her case har stadig mere end rigelig kompleksitet til at software design er relevant.
- Jeg håber, at det er lidt sjovt, at det er et spil, i skal lave, istedet for at i skal lave en "Møntboks 2".