Versionsstyring i programmerings-undervisningen

Versionsstyring kan gøre programmeringsprocessen mere systematisk ved at gemme versioner af kode efterhånden den ændres. Her gives en introduktion til konceptet, værktøjer til implementering og avendelser i undervisningen.

Versioner af kode

Computerprogrammer udvikles iterativt og hver iteration af udviklingsprocessen resulterer i en ny version koden og programmet.

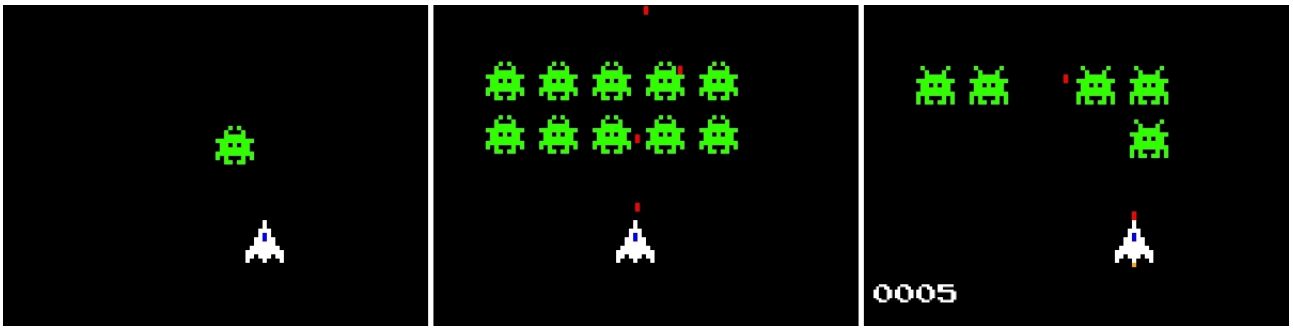

Fig. 1 Tre versioner af et generisk arkadespil, Space Shooter. Animation tilgængelig her: https://jonascj.dk/emu-versionsstyring/

Desværre gemmes ofte kun nyeste version af koden. Gamle versioner forkastes mere eller mindre bevidst ved filen eller filerne, som indeholder koden til programmet, overskrives når nyeste version af koden gemmes. Alle versioner af koden skal ikke gemmes, men fordelene ved at gemme udvalgte versioner af koden er mange:

- En fungerende udgave af programmet eksisterer altid, selv mens du arbejder på at tilføje ny funktionalitet som endnu ikke virker.
- Elever kan f.eks. demonstrere deres program selvom de arbejder på noget som p.t. får programmet til at fejle.
- Et mislykket eksperiment med koden kan fortrydes, selvom editorens fortryd-funktion ikke kan fortryde så mange ændringer (f.eks. 3 timers arbejde fordelt over 2 dage).
- Dokumentation af udviklingsprocessen gøres lettere: historiske versioner af koden kan findes frem og screenshots af historiske versioner af programmet kan produceres når eleverne skal skrive synopsis over deres eksamensprojekt.
- En feature eller en implementering som fandtes i version 1 af programmet, men som blev fjernet i version 2 af programmet, kan findes frem igen og blive en del af version 3 af programmet igen.

Undertiden gemmes versioner af simple projekter/programmer som illustreret på figur 2. Hver gang programmet har nået en milepæl, eller før et eksperiment påbegyndes, gemmes en kopi af koden i en fil med et nyt filnavn.

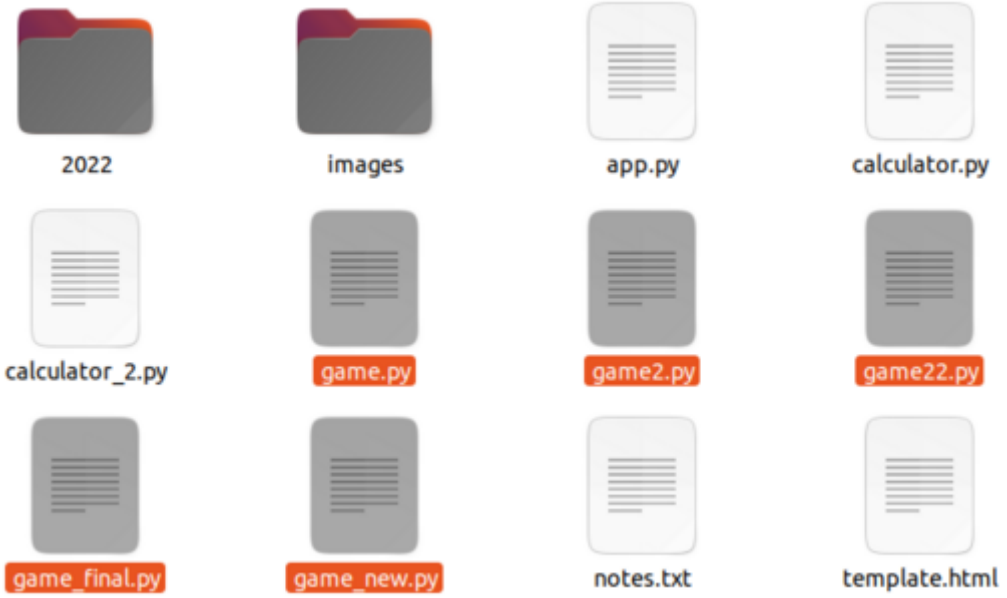

Fig. 2 Mappe med bl.a. Python-filer som kunne udgøre fem forskellige versioner af Space Shooter fra figur 1.

En sådan løsning er bedre end ingen løsning, men der er åbenlyse svagheder ved løsningen:

1. Det er svært at se hvilken fil som indeholder nyeste version af koden (filnavn og evt. tid/dato for redigering kan dog give en indikation).
2. Det er svært at gennemskue hvorfor en given version af koden er gemt, kun filnavnet indeholder et hint om årsagen.
3. Hvis projektet består af mange filer, f.eks. et spil med lydfiler, billeder og kode, forværres 1. og 2. typisk.

Derfor findes mere forfinede og avancerede systemer til *versionsstyring*.

Versionsstyring

Systemer/værktøjer til **versionsstyring** (engelsk: version control) systematiserer processen med at gemme og annoterer versioner af filer som udgør et softwareprojekt. Figur 3 viser forskellige versioner af Space Shooter-projektet som er versionsstyret med værktøjet *Git*. En version er et snapshot af alle projektets filer som de så ud på et givent tidspunkt, i Git-terminologi kaldet et *commit*[1].

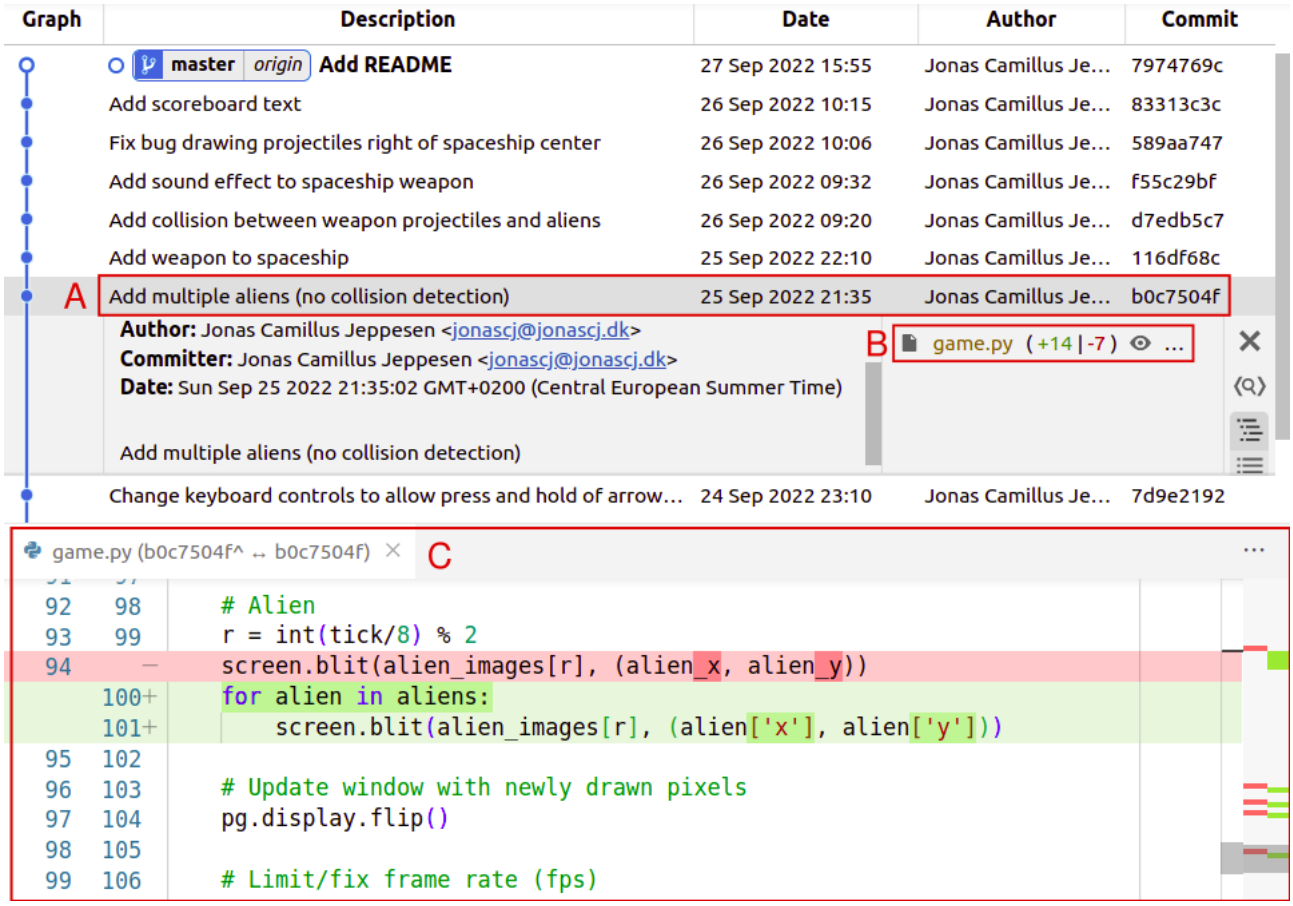

Fig. 3 Forskellige versioner af et projekt versionsstyret med Git og visualiseret vha. Visual Studio Code med udvidelsen Git Graph.

- **(A)** Ét blandt en række *commits* i projektet (versioner af projektet). Dette commit har navnet b0c7.

  Bemærk beskrivelsen af hvert commit (commit-beskeden) f.eks. `Add multiple Aliens (no collision detection)`, som forklarer årsagen til en ændring.

- **(B)** Detaljeret indblik i commit b0c7 som bestod af ændringer til filen `game.py` (14 linjer tilføjet, 7 linjer fjernet).

- **(C)** Visualisering af ændringer som er sket i `game.py` fra forrige version (7d9e) til denne version (b0c7).

Linjer overstreget med rød er blevet fjernet og linjer overstreget med grøn tilføjet, f.eks. er linje 94 `screen.blit(alien ...` blevet erstattet af linje 100-101 som indeholder et for-loop.

Git

Det mest benyttede system/værktøj til versionsstyring er Git[2] [3]som også har lagt navn til platformen GitHub. Git er gratis, open source, og virker under både Windows, macOS og Linux.

Git er i udgangspunktet et kommandolinjeværktøj, men Git er også integreret i flere moderne udviklingsmiljøers grafiske brugerflade, hvilket gør det tilgængeligt for elever. Følgende afsnit giver et overblik over Git og hvordan Git kan bruges i undervisningen via Visual Studio Code (VS Code), et af de mest populære udviklingsmiljøer[4].

En detaljeret tutorial i installation og brug af Git via VS Code er uden for rammerne af denne artikel, men kan findes her: https://jonascj.dk/git.

Repositories og projektstruktur

En mappe med filer som er versionsstyret af Git kaldes et **repository**. Ét projekt har ét repository som opbevarer projektets filer og deres forskellige versioner. Figur 4 viser mappen med alle filer som udgør spillet Space Shooter fra figur 1 og den skjulte mappe `.git` som gør mappen til et Git-repository.

Git opbevarer information om projektets historik (commits) i `.git`-mappen. Alle som har en kopi af `.git` mappen fra figur 4 har en fuld kopi af Space Shooter's repository og kan interagere med det uden at være i forbindelse med en server via f.eks. en internetforbindelse.

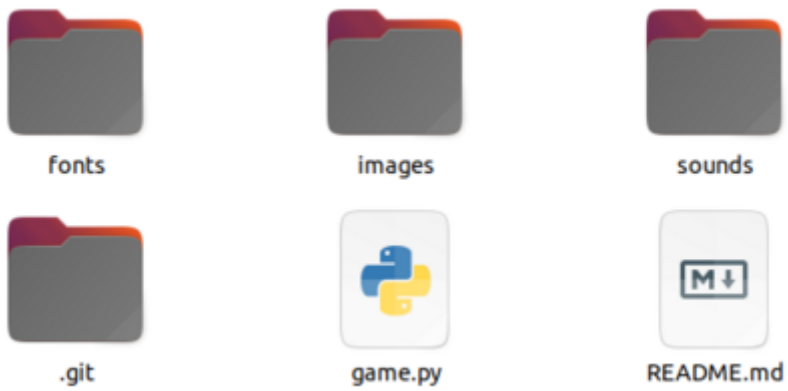

Fig. 4 Mappen `space-shooter-python-git` som udgør spillets repository lokalt på en computer.

Når der udvikles på projektet foretages ændringer i filerne fra figur 4 som kaldes repository'ets **working tree**. I VS Code er mappen som udgør projektets repository åben og repository'ets working tree er synligt i VS Codes *Explorer*-panel, som vist på figur 5.

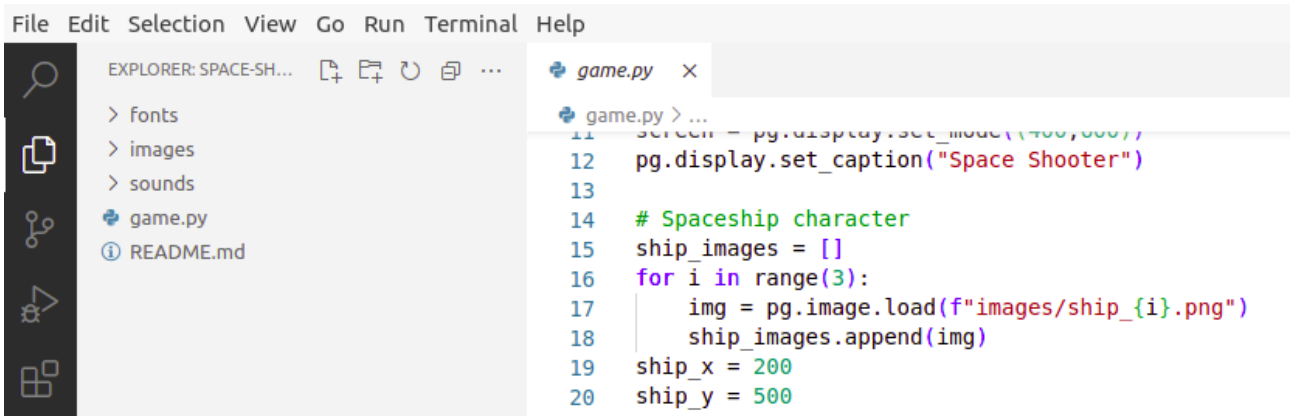

Fig. 5 Mappe (repository) åben i VS Codes Explorer-panel.

En kopi af Space Shooter's lokale repository er også hostet hos GitHub: https://github.com/jonascj/space-shooter-python-git. Efterhånden som der laves nye commits det lokale repository kan GitHub's kopi synkroniseres med det lokale repository. GitHub kan således fungere som en form for backup af et projekts repository. Hvis der er flere udviklere på et projekt, f.eks. to elever, kan GitHub også facilitere samarbejde ved at lade udviklerne udveksle opdateringer til projektets repository gennem GitHub.

Man behøver ikke bruge Git for at anbefale sine elever at benytte en mappe pr. projekt, men med Git bliver man tvunget til det. På den måde undgås situationen skildret på figur 2, at flere projekter blandes i en mappe, hvilket uhensigtsmæssigt og uoverskueligt.

Opret eller klon repository

En vilkårlig mappe med filer kan konverteres til et Git-repository. VS Code's *Source Control*-panel tilbyder oprette et repository i en åben mappe hvis mappen ikke allerede er et repository, som vist på figur 6.

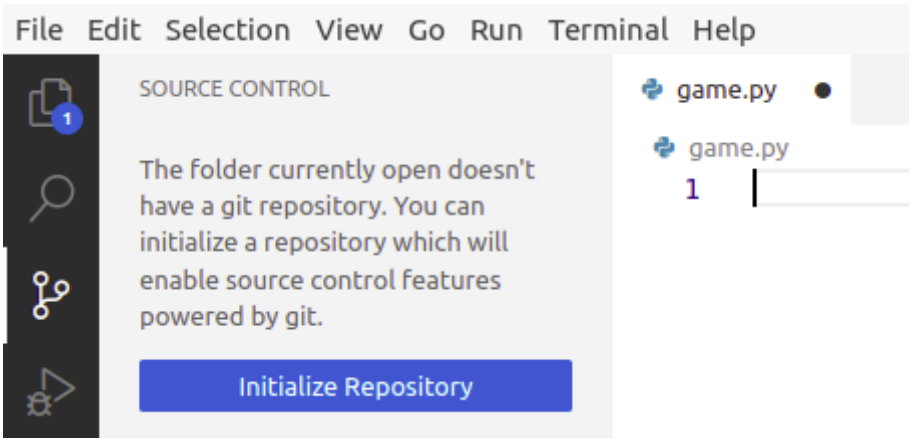

Fig. 6 Oprettelse af repository via VS Code's Source Control-panel.

Repositories kan også klones fra et andet repository, f.eks. fra et repository hostet på GitHub. Åbnes et nyt vindue i VS Code, således ingen mappe er åben i VS Code, tilbyder VS Code's *Source Control*-panel at klone et repository som vist på figur 7. Elevernes første handling med Git kan passende være at klone det repository (projekt) de skal arbejde videre på.

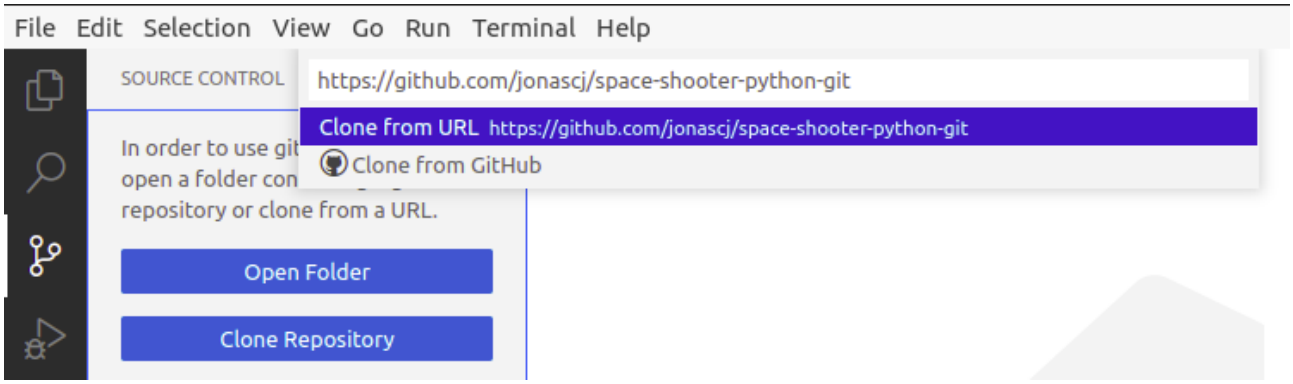

Fig. 7 Kloning af repository via VS Code's Source Control-panel.

Repository-status

Når der ændres i et projekts filer ændres status projektets repository. Nuværende status af et repository ses i VS Code's *Source Control*-panel. Figur 8 viser status efter at have ændret animationen af rumskibet fra at bestå af 3 billeder til at bestå af 4 billeder. Klikkes på `game.py` i oversigten over repository-ændringer vises ændringerne foretaget i filen siden sidste commit: linje 16 markeret med rød er blevet ændret til linje 16 markeret med grøn. Visningen hvor ændringer til en given fil er markeret linje for linje kaldes undertiden et **diff** (difference). Ændringer kan let fortrydes ved at klikke på Discard Changes ud for en given fil.

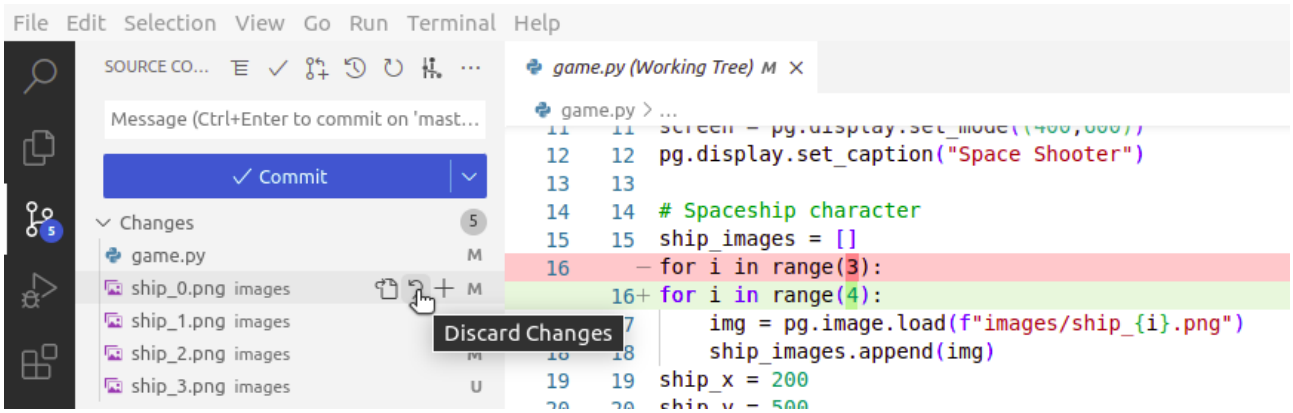

*Fig. 8 Repository-status efter at have redigeret 3 billeder, tilføjet 1 nyt billede og ændret i koden i `game.py`. Filer markeret med **M** modificerede og filer markeret med **U** er endnu ikke tilføjet repository'et (untracked).*

Elever bringer ofte koden til et program i en tilstand hvor programmet ikke længere er funktionelt eller hvor programmet slet ikke kan afvikles. I sådanne tilfælde er det yderst værdifuldt at kunne identificere ændringer i forhold til sidste commit hvor programmet virkede.

Skyldes situationen en utilsigtet ændring af koden fortrydes ændringen blot med pilen **Discard Changes**. Er situationen derimod resultatet af et forsøg på at implementere noget nyt i programmet, kan diff-visningen fra figur 8 gøre det lettere for eleven at forklare sine ændringer og lettere for underviseren at hjælpe eleven videre.

Commits

Når arbejdet med en ændring af projektet er færdigt, f.eks. når ny funktionalitet er implementeret og testet, laves et nyt commit inden nyt arbejde påbegyndes. Nye commits laves fra VS Code's Source Control-panel (fig. 9).

Ændringerne fra forrige afsnit kan tilføjes projektet som et commit ved at trykke på knappen **Stage Changes** ud for ændring som ønskes medtaget i det kommende commit, skrive en commit-besked (`Change spaceship animation`) og trykke **Commit**. Det nye commit (64ee) er nu synligt i Git's log (historik), som vist på figur 10.

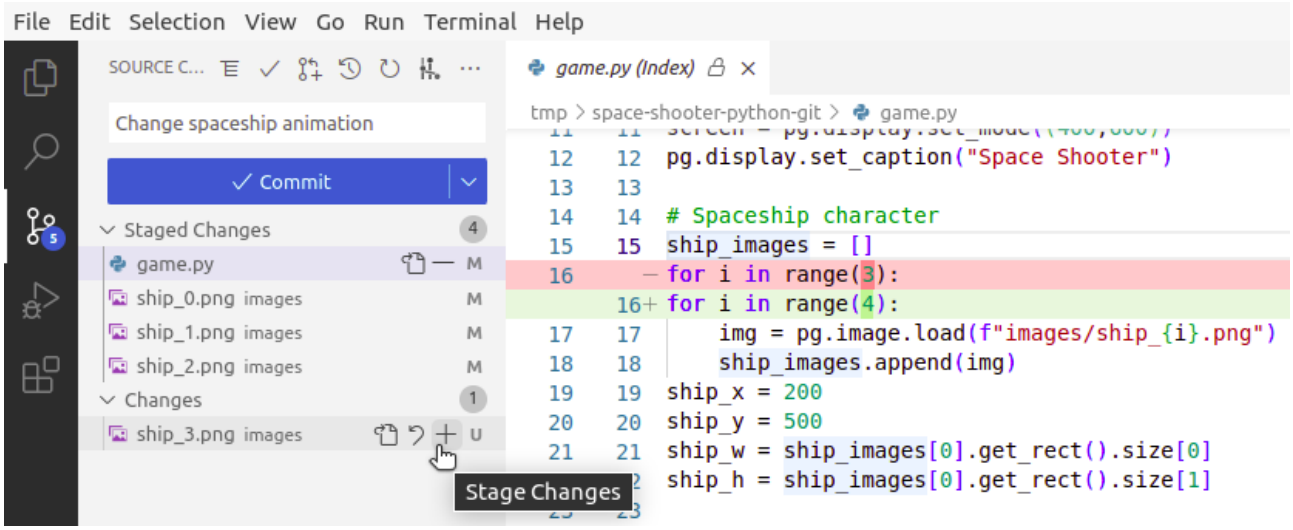

Fig. 9 Oprettelse af nyt commit via VS Code's Source Control-panel.

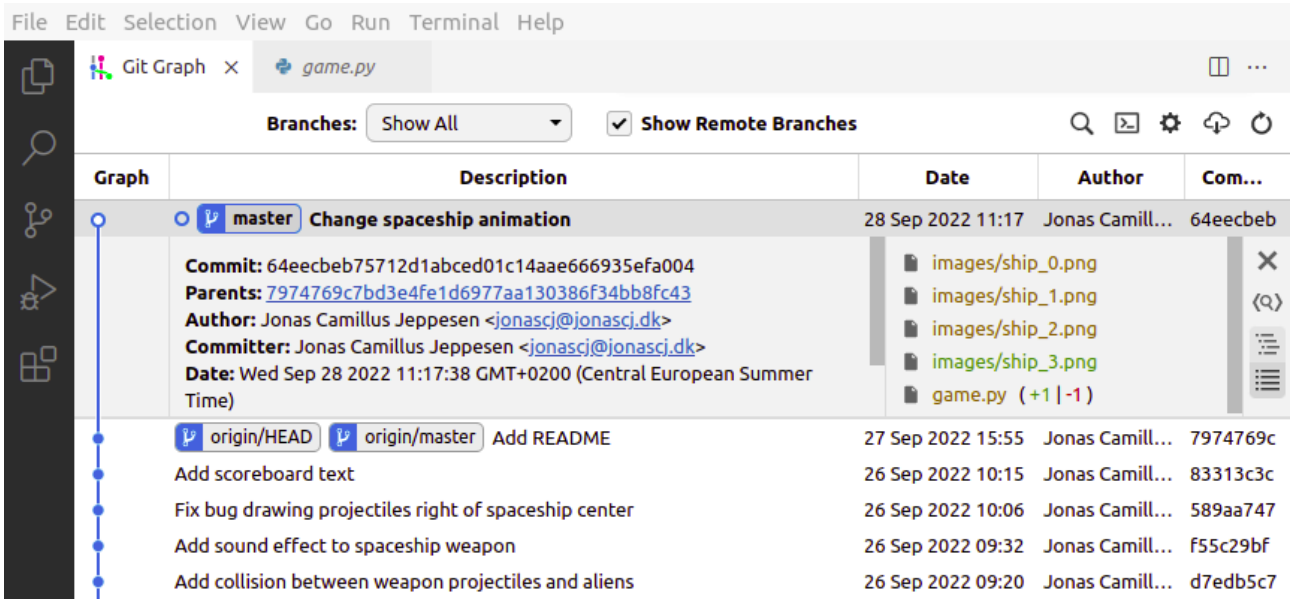

Fig. 10 Git's log som præsenteret af Git Graph (en udvidelse til VS Code).

Skift mellem commits

Fordelen ved at gemme udvalgte versioner af et projekt i form af commits er, at Git let kan skift mellem dem. Fra Git Graph-visningen kan laves et såkaldt **checkout** af et vilkårligt commit (se figur 11). Ved et checkout opdateres et repository's working tree til at se ud som det gjorde da pågældende commit blev lavet. Explorer-panelet på figur 11 viser nu de filer som udgjorde projektet da commit a253 blev lavet.

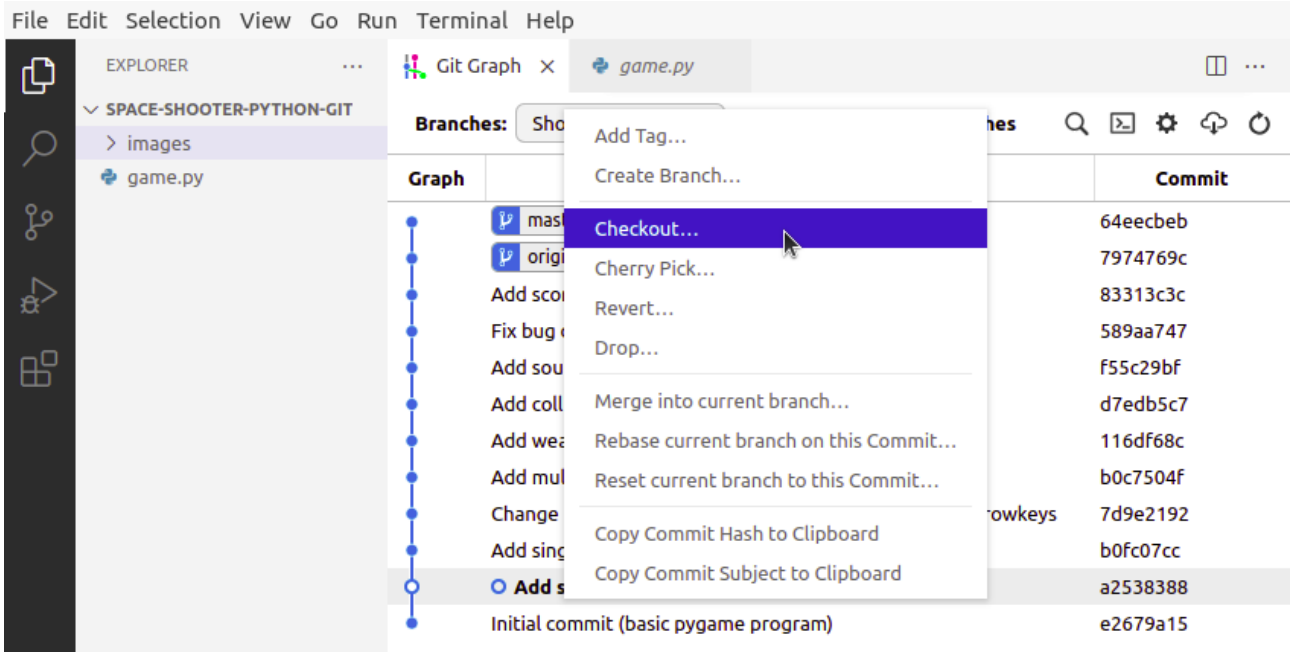

Fig. 11 Git Graph-visningen kan bruges til at skifte mellem commits. Bemærk mapperne `fonts` *og* `sounds` *samt filen* `README.md` *ikke er at se i Explorer-panelet (som de er på figur 4 og 5). De eksisterede ikke da commit a253 blev lavet.*

Med evnen til at skifte mellem commits kommer fordelene nævnt i det indledende afsnit, f.eks. kan eleverne finde gamle versioner af koden frem når de skal skrive synopsis over deres eksamensprojekt.

Hvis kode udleveres til elever i form af et Git-repository, udleveres også kodens udviklingshistorie som eleverne ellers sjældent ser[5]. Eleverne kan gennem checkouts af forskellige commits undersøge kodens udvikling og se et eksempel på *›stepwise improvement‹* i praksis.

Branch og stash

En programmør kan have brug for midlertidigt at lade sit projekt forgrene sig[6] og dette understøtter Git med **branches**. Alle commits omtalt og vist indtil nu har tilhørt en branch kaldet *master*, men det er muligt at have flere branches, som hver kan indeholde sine egne commits (versioner af projektet).

Inden arbejdet med en ny feature påbegyndes kan en ny branch med fordel oprettes til arbejdet. Det er nemlig ikke sikkert den kode som bliver skrevet, og gemt i et eller flere commits, skal bruges eller kan bruges:

- det kan mislykkes at implementere den nye feature,
- den nye feature kan blive uønsket mens arbejdet står på,
- eller det kan blive nødvendigt midlertidigt at arbejde en anden feature.

Figur 12 viser et alternativt udviklingsforløb for Space Shooter-spillet som benytter branches. Efter at have tilføjet et våben til rumskibet i commit `f6f8` arbejdes der for en tid på at tilføje to våben til rumskibet i en særskilt branch med navn `dual-weapons`. Arbejdet forkastes eller efterlades imidlertid, måske fordi to våben alligevel ikke var tiltalende, og udviklingen fortsætter i *master*. Branch'en `dual-weapons` slettes ikke, så arbejdet kan potentielt set fortsættes senere. Arbejdet i branch'en `sound-effects` integreres (engelsk: **merge**) derimod i arbejdet udført i *master*. Et **merge** kan ofte håndteres automatisk af Git, men der kan være konflikter (engelsk: **merge conlicts**) som skal løses manuelt.

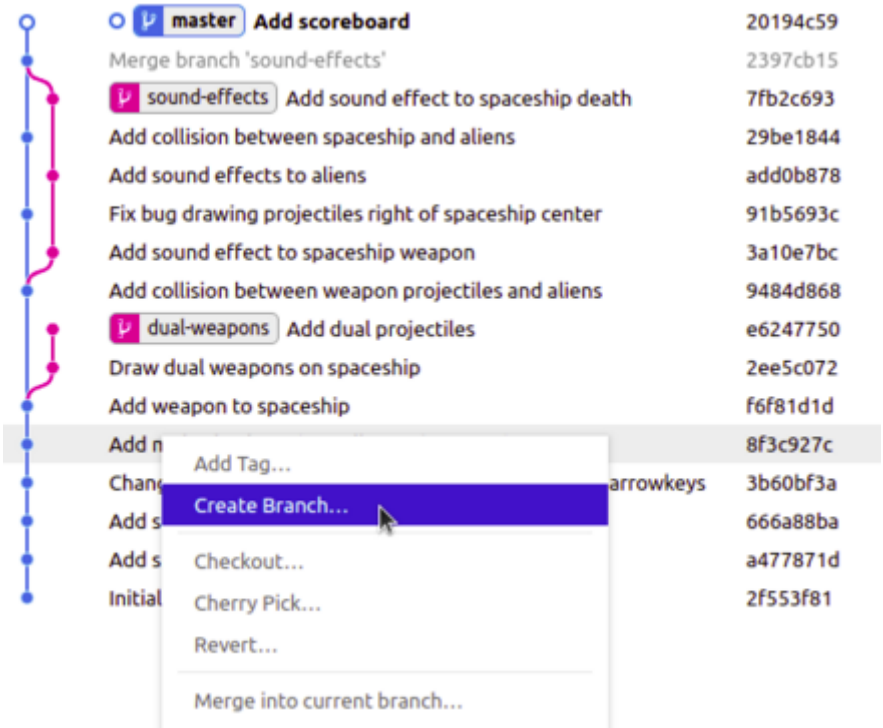

Fig. 12 Log og graf for et repository med 3 branches: `master`*,* `dual-weapons` *og* `sound-effects`*.*

Det er let at lave branches (højreklik på et commit og vælg **Create Branch…**) og det er let at skifte mellem branches (dobbeltklik på et branch-navn). Hvis en elev går i stå med en given opgave, og det har lange udsigter at få hjælp, kan eleven lave en ny branch og arbejde på noget andet.

Dog er det ikke muligt at skifte branch, hvis et repository's working tree indeholder ændringer som ikke er committed (et såkaldt **dirty** working tree). Eventuelle ændringer kan gemmes midlertidigt Git's **stash** før der skiftes branch og kaldes frem når der skiftes tilbage.

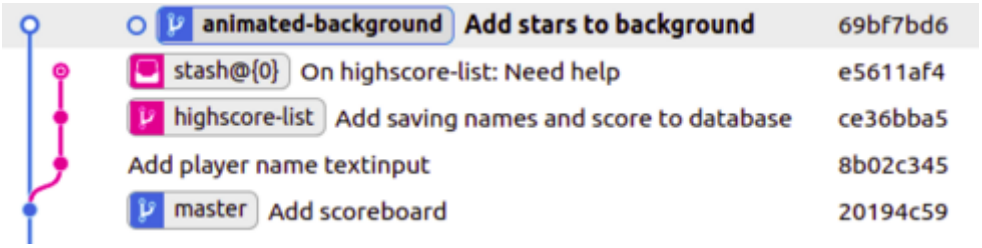

Fig. 13 Commit e561 er et stash som gemmer ændringerne der blev arbejdet på i branch'en highscore-list da det blev besluttet skifte branch for at arbejde på noget andet.

GitHub

Benyttes GitHub som såkaldt **remote repository**, hvor en kopi af et lokalt repository hostes hos GitHub, giver det mulighed for en række aktiviteter i undervisningen:

- Elever synes typisk det er spændende at dokumentere deres projekt i den `README.md`-fil som vises på forsiden af GitHub's visning af et repository.
- Elever kan nemt klone hinandens projekter med henblik på at lave peer-review af kode eller at agere testere for hinanden.

Eleverne kan eventuelt, efter en peer-review aktivitet, oprette **issues** i hinandens GitHub-repositories. Det træner elevernes skriftlige formidling af tekniske emner.

- Kode kan, som tidligere nævnt, udleveres til elever via GitHub, selvom et repository lige så vel kan udleveres som en ZIP-fil.
- GitHub kan på frivillig basis bruges som logbog. Elever kan opretter et repository pr. projekt/opgave og dokumenterer koden i en `README.md`-fil, som kan indeholder både figurer, screenshots mm.

GitHub skal selvfølgelig betragtes som enhver anden online service elever kan anbefales at benytte frivilligt[7]. Eleverne skal oplyses konsekvenserne ved at benytte GitHub's service; repositories som udgangspunkt offentligt tilgængelige og GitHub må benytte brugernes kode til at forbedre deres produkt[8] (f.eks. træne maskinlæringsmodeller[9]). Eleverne kan anbefales at benytte en mailadresse stillet til rådighed af skolen når de skal oprette en bruger.

Der findes også alternativer til GitHub, f.eks. GitLab som kan hostes af skolen eller underviseren selv.

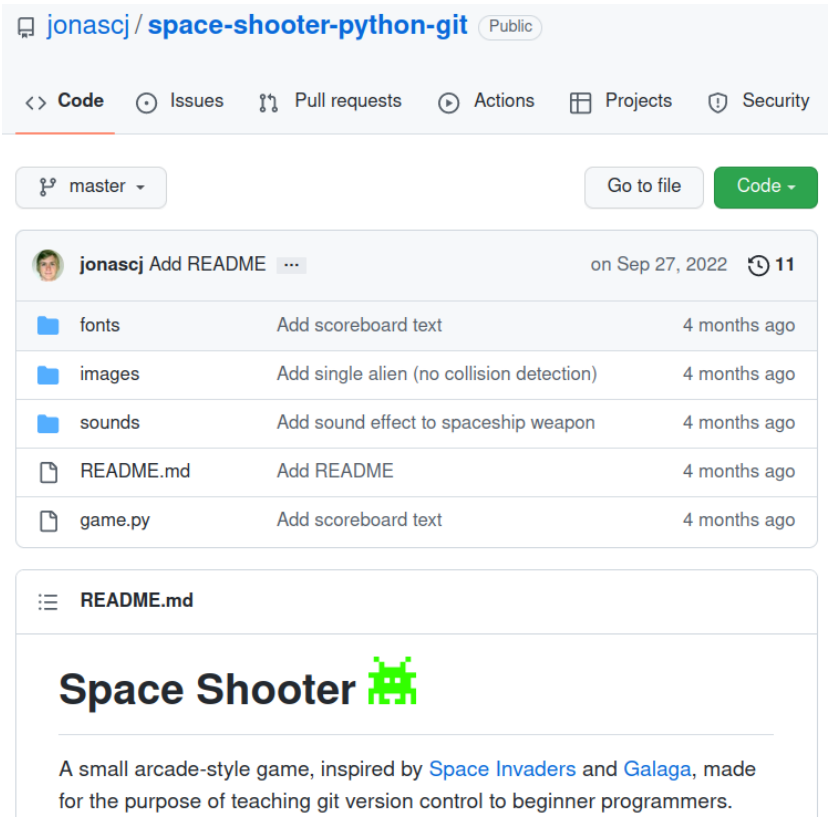

Fig. 14 GitHub's visning af Space Shooter's repository.

Didaktiske overvejelser

Kan man lære versionsstyring af kode og programmeringsprojekter samtidig med man lærer at programmere? Det spørgsmål bør man stille sig selv før man beslutter at introducere elever til versionsstyring. Et værktøj som Git har intet med programmering at gøre, computerprogrammer kan sagtens udarbejdes uden brug af versionsstyring og Git kan lige såvel bruges til versionsstyring af manuskripter som kode.

I mange tilfælde kan versionsstyring dog gøre programmeringsprocessen mere systematisk og facilitere arbejdet med at skrive kode. Hvorfor bruge tid på at undre sig over hvilken del af koden der er blevet ændret (således programmet nu ikke virker) når Git kan besvare det spørgsmål hurtigt og præcist? Hvorfor smide fungerende versioner af sin kode væk, når de let kan gemmes som commits i et Git-repository?

Brug af versionsstyring i programmeringsundervisningen kan således bidrage til at opfylde det faglige mål som omhandler systematik i programmeringsprocessen, men det kræver også introduktion af et nyt værktøj og nye koncepter. Introduceres det for tidligt, risikerer eleverne at bruge for meget tid på at tænke over versionsstyring og for lidt tid på at øve sig i at skrive kode. Introduceres det for sent stifter eleverne kun bekendtskab med konceptet, men opnår ikke erfaring med værktøjet og får ikke gavn af værktøjet.

Ifølge vejledningen til læreplanen er det vigtigt, at elever med programmering på B-niveau lærer metoder til at samarbejde om udviklingen af programmer/software. Git's branch- og merge-funktionalitet kan sammen med GitHub facilitere samarbejde, men det er sværere at arbejde

sammen via Git end at arbejde solo med Git. Det taler for at eleverne tidligt introduceres til solo-arbejde med Git, for senere at blive introduceret til samarbejde via Git.

Praktiske erfaringer

I praksis har jeg introduceret klasser med programmering på B-niveau til Git efter ét års programmeringsundervisning (halvejs i fagets forløb).

På det tidspunkt oplever jeg, at mange fagligt stærke elever tager Git til sig som værktøj og begynder at bruge det i deres daglige arbejde. Denne elevgruppe kunne meget muligt have gavn af at blive introduceret til Git tidligere, måske efter 3–6 måneders undervisning.

Det store midtersegment af fagligt middelstærke elever benytter Git i forbindelse med det forløb som introducerer Git og finder det meningsfuldt, men benytter det efterfølgende ikke. Enkelte middelstærke elever genoptager brugen af Git til deres eksamensprojekt, når de mindes dem om de skal dokumentere udviklingsprocessen og ikke kun det færdige produkt.

Fagligt svage elever har jeg ikke oplevet bruge Git i betragteligt omfang. Når jeg har introduceret Git efter ét års undervisning, har denne elevgruppe ofte faglige udfordringer som gør det svært for dem at fokusere på at lære et nyt værktøj. Ofte har jeg vurderet værktøjet ikke vil gavne denne elevgruppe og givet dem lov til ikke at sætte sig ind i dets brug.

Introduktionsforløbet har typisk bestået i at lade eleverne klone et repository som indeholder et program/projekt af en type de allerede er bekendt med, f.eks. https://github.com/jonascj/space-shooter-python-git hvis eleverne allerede er bekendt med spilprogrammering. Det er nemmere at lære et nyt værktøj at kende, hvis man ikke samtidig skal lære nye biblioteker, strukturer og teknikker.

Næste gang jeg får mulighed for at introducere en klasse til Git vil jeg forsøge at gøre det væsentligt tidligere end jeg har gjort hidtil. Ved at introducere Git tidligere kan det gøres mere gradvist og eleverne kan potentielt set have gavn af værktøjet i længere tid.

Litteratur

[1] Git development team. A git glossary. 2022. URL: https://git-scm.com/docs/gitglossary.

[2] Stack Overflow. Developer survey - source control. 2015. URL: https://insights.stackoverflow.com/survey/2015#tech-sourcecontrol.

[3] Stack Overflow. Developer survey - version control system. 2022. URL: https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems.

[4] Stack Overflow. Developer survey - integrated development environment. 2022. URL: https://survey.stackoverflow.co/2022/#section-most-popular-technologies-integrated-development-environment.

[5] Michael E. Caspersen. *Educating Novices in the Skills of Programming*, pages 5–6. ph.d.-afhandling, 2007. URL: https://www.cs.au.dk/~mec/dissertation/Dissertation.pdf.

[1] Git development team. A git glossary. 2022. URL: https://git-scm.com/docs/gitglossary.

[6] Ben Straub Scott Chacon. Progit - 3.2 git branching - basic branching and merging. 2022. URL: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging.

[7] Styrelsen for It og Læring for Børne- og Undervisningsministeriet. Vejledning til sikker brug af sociale medier og digitale værktøjer i undervisning. 2021. URL: https://emu.dk/sites/default/files/2021-08/Vejledning%20til%20sikker%20brug%20af%20sociale%20medier%20og%20digitale%20v%C3%A6rkt%C3%B8jer%20i%20undervisning_ungdomsuddannelser.pdf.

[8] GitHub Terms of Service. License Grant to Us. URL: https://docs.github.com/en/site-policy/github-terms/github-terms-of-service?ref=fossa.com#4-license-grant-to-us

[9] Reuters. OpenAI, Microsoft want court to toss lawsuit accusing them of abusing open-source code. URL: https://www.reuters.com/legal/litigation/openai-microsoft-want-court-toss-lawsuit-accusing-them-abusing-open-source-code-2023-01-27/