Mittuniversitetet

MID SWEDEN UNIVERSITY

Metoder och verktyg i mjukvaruprojekt

Projekt – Game of Life

Projektet syftar till att studenter får tillämpa kunskaperna från kursens laborationer i ett något större mjukvaruprojekt!

Innehållsförteckning

1 Game of Life

Game of Life [1] uppfanns 1970 av matematikern John Conway och är ett exempel på cellulär automata som i det här fallet innebär ett 2-dimensionellt rutmönster uppbyggt av celler, som sammantagna utgör simuleringens population. I den klassiska implementationen kan varje cell befinna sig i tillståndet **på** eller **av** (*levande* eller *död*). Simulationen startar i ett initialt tillstånd och fortlöper genom att applicera enkla regler för att ta populationen till nästa generation. De klassiska reglerna är:

- En levande cell som har färre än två levande grannar dör av under-population.
- En levande cell som har två till tre levande grannar lever vidare till nästa generation.
- En levande cell som har fler än tre levande grannar dör av över-population.
- En död cell som har exakt tre levande grannar börjar leva (reproduktion).

För att generera nästa generation spelar det ingen roll vilken ordning man applicerar reglerna på. Viktigt är dock att man skapar nästa generation i ett skilt minnesutrymme från den generation som man utgår från, d.v.s. att man läser grannar från den nuvarande generationen och skriver cellernas nya tillstånd till minnesblocket som representerar nästa generation.

1.1 Applikationen

I tidigare kursomgångar var kravet att studenterna skulle göra sin egen implementation av *Game of Life* baserat på en mängd olika inkluderingskriterier. I senare upplagor har fokus flyttats från konstruktionen till att istället innefatta de moment gällande metoder och verktyg som behandlas i kursens moduler. Därmed får ni tillgång till en fungerande implementation av *Game of Life* som sedan utgör utgångspunkten för det fortsatta arbetet.

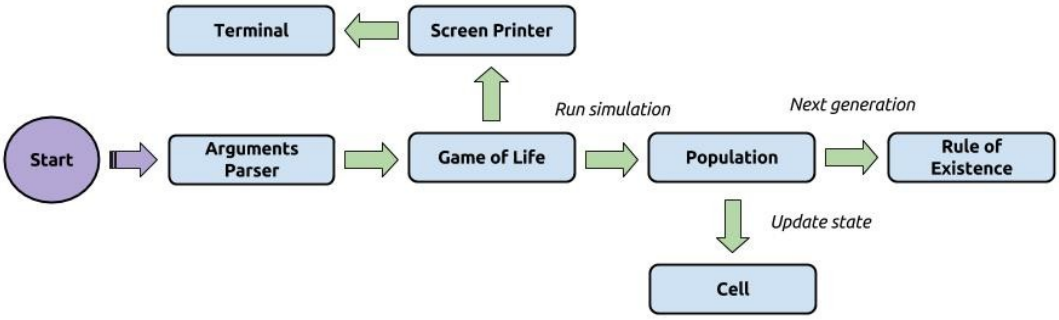
Kompileringen producerar en exekverbar fil vid namn **`GameOfLife`**. Exempel på exekvering med och utan argument:

```
GameOfLife
GameOfLife -s 80x24 -g 200
GameOfLife -s 160x24 -er erik
GameOfLife -s 160x24 -er conway -or erik -g 1000 -f
Population_Seed.txt
```

Kom ihåg att i **GitBash** används syntaxen; **`./GameOfLife.exe ...`**

MÖJLIGA INPARAMETRAR	
`-h`	Visar hjälpinformation och avslutar sedan applikationen direkt.
`-s`	Används för att ange storlek på den simulerade världen i form av bredd x höjd. Default är **`80x24`**
`-f`	Används för att läsa in en fil som anger starttillstånd på cellpopulationen samt dess dimensioner (ersätter **-s**). Om parametern inte används kommer starttillståndet slumpas fram.
`-g`	Anger hur många generationer simulatorn skall iterera över innan programmet avslutas.
`-er`	Anger regel som skall användas för jämna generationer. Default är **`conway`**
`-or`	Anger regel som skall användas för udda generationer. Default är samma som **`-er`**

Programmet följer denna exekveringsordning;



Det första som sker vid programstart är att eventuella inparametrar hanteras och används för att ange förutsättningarna för den fortsatta simuleringen. Därefter tar en instans av **`GameOfLife`** över och fungerar som ett slags mellanled mellan simuleringens interna värden och dess visuella presentation på skärmen. **`Population`** representerar den totala mängden celler som utgör simuleringens tvådimensionella värld, där tillståndet för varje **`Cell`** bestäms utifrån särskilda regler. Det är mycket enkelt att lägga till fler regler än de som redan finns, och ni får gärna bidra med egna!

Detta var en mycket kort presentation av programmets väsentliga delar, och större förståelse skall ni erhålla genom att studera programkoden!

> *Under exekveringen av applikationen kan ingen kommunikation ske från användaren. Simuleringen startar utifrån angivna värden, itererar igenom alla generationer och kontinuerligt uppdaterar skärmen. När simuleringen nått sista generationen kommer programmet avslutas!*

2 Uppgift

Projektuppgiften skall bedrivas i samarbete med en kurskamrat, och får således inte utföras enskilt eller i grupper om fler är två personer. I första hand formar ni dessa grupper själva genom att diskutera inom studentgruppen, men vid problem hanterar kursansvarig denna indelning med förbehållen rätt att eventuellt utfärda undantag beträffande enskilt arbete (tillsammans med eventuell anpassning). Under kursmomentet i Moodle finns ett verktyg för gruppvalet samt en mängd förskapade grupper, som alla initialt saknar medlemmar. Högsta medlemsantal för grupperna är två och ni använder gruppverktyget för att ange er tillhörighet. Allteftersom dessa grupper fylls kommer respektive medlem erhålla access till avsett grupprepo i kursens Bitbucket Team, och det är detta repo som skall användas för projektarbetet.

Basrepot för projektet finner ni i kursens **Bitbucket Team** och det innehåller allt ni behöver för att genomföra arbetet. Klona detta repo och ange erat tilldelade grupprepo som ny **`remote origin`**:

```
$ git clone --recursive git@bitbucket.org:miun_dt042g/gof_project.git
$ git remote remove origin
$ git remote add origin
git@bitbucket.org:miun_dt042g/group_X_vt21_project.git
```

Eftersom ni arbetar i par är det enbart en av er som skall utföra detta moment. Därefter kan varje gruppmedlem klona detta repo till sitt lokala system, precis som i tidigare moment. Kom dock ihåg att ange flaggan **`--recursive`** då **Terminal** ingår som **submodule**!

Programkoden för *Game of Life* är mycket svagt dokumenterad och det blir er uppgift att studera denna för att förstå dess implementation. Utifrån denna förståelse skall ni sedan upprätta enhetstester och dokumentation genom ett agilt arbetsflöde. Nedan finner ni detaljer kring kraven för dessa olika delar!

> *Utför löpande commits alltigenom arbetet tillsammans med beskrivande meddelanden. Det skall vara möjligt att få en överskådlig bild av arbetet utifrån denna historik!*
>
> **`.gitignore`** *skall även den löpande uppdateras med nödvändig information, allteftersom behov uppstår!*

2.1 Utvecklingsmetod

Utvecklingen för arbetet skall följa den agila metoden **Kanban** som planeras och administreras via en **Trello board**. Ni skall gemensamt komma överens om en lämplig *branchstrategi* som skall användas, men det måste understödja ett granskande moment innan branchen sammanförs med huvudgrenen. Ni skall alltså använda ***peer review*** för att godkänna bidrag. Ett lämpligt arbetsflöde som stödjer detta är **Feature Branch Workflow** [3] som dessutom smidigt kan användas tillsammans med *Bitbucket* för att skapa s.k. **pull requests** [4].

Ni skall skapa en Trello board för erat team där arbetet kan planeras och fördelas. Utgå från följande **Kanban Exempel** och skapa en motsvarighet för teamet där alla listor skall finnas med;

- `Manifest`
- `Backlog`
- `In Progress`
- `Review`
- `Completed`

Detaljer för struktur och innehåll läser ni i de kort som finns i exemplets manifestlista, och det är viktigt att ni studerar dessas innehåll eftersom denna metadata anger förutsättningar och tillvägagångssätt för såväl samarbete som övriga utvecklingsdetaljer. Ni kommer senare att bedömas utifrån hur väl ni förhåller er till presenterade kriterier gällande;

- `Git workflow`
- `Task Guidelines`
- `Peer Review Guidelines`

Det är helt ok om ni väljer att kopiera dessa detaljer från exempelmanifestet, men ni kan också upprätta egna kriterier. Det viktiga här är att ni konsekvent följer de definierade kraven inom dessa delar!

> *I Trello-exemplet finns även några **tasks** som visar hur arbetsuppgifter kan utformas. Tänk dock på att varje task skall representera en relaterad commit för att granskningskontrollen skall kunna administreras. Lämpligen använder ni namnet på arbetsuppgiften som **commit message** och i Trellokortet kan ni länka till den specifika committen.*
>
> *Tänk även på omfattningen av varje arbetsuppgift som bör vara mycket begränsad och tydligt avgränsad till ett specifikt område. Dels blir översikten i Trello tydligare och administrationen av arbetsuppgifter enklare av detta, men även versionshistoriken blir faktiskt mer överskådlig med många men små bidrag.*

2.2 Enhetstester

Ni skall upprätta lämpliga **enhetstester** med ramverket **Catch** som speglar programmets specifikationer gällande såväl moduler som procedurer. Integriteten av både klasser och funktioner skall således kunna verifieras genom dessa tester, där eventuella svagheter och felaktigheter framgår. ***Det är fritt fram att modifiera projektets gränssnitt för att möjliggöra de tester som behövs, men inte för att åtgärda buggar funna i implementationen!***

Kompileringen skall producera en exekverbar fil som heter `GameOfLife-Test` som vid körning utan parametrar exekverar alla enhetstester. Byggskriptet behöver således garantera skapandet av minst två *run configurations*, en med tester samt en utan!

2.3 Dokumentation

Programkoden skall utförligt kommenteras i enlighet med de konventioner som behandlas i *lektion 5* och dokumentation (**HTML**) skall genereras med hjälp av **Doxygen** där varje deltagare skall kunna skapa denna utifrån en gemensam `doxyfile`, som placeras under `Docs/` i projektet. Den genererade dokumentationen skall vidare ge en bra överblick av projektet och dess ingående entiteter. Tänk även på att utesluta denna **HTML**-output från versionshanteringen!

Kommentarer för att generera dokumentationen skall presenteras i deklarationerna (header) av respektive entitet och tydligt redogöra aktuellt ansvarsområde. För både klasser och funktioner skall taggarna `@brief`, `@details` samt `@test` användas, och funktioner skall även redogöra för eventuella inparametrar samt returvärden. Eventuella avvikelser från specifikationen som testerna har funnit skall återges via taggen `@bug` i anslutning till aktuell modul / procedur.

Den genererade **HTML**-dokumentationen skall innehålla en `Main Page` som kortfattat beskriver projektet tillsammans med en bild. Vad denna bild föreställer spelar mindre roll men viktigt är att den, tillsammans med andra eventuella bilder, placeras under `Docs/img/` och att relativa sökvägar anges i konfigurationsfilen. Dessutom skall det finnas en fristående sida där gruppens medlemmar presenteras i separata sektioner (`@section`). Länkning mellan `Main Page` och denna presentationssida skall ske genom taggen `@ref`!

3 Redovisning

Tillför en **`README.md`** under projektroten med information som beskriver projektet samt hur arbetet har genomförts. Minst följande rubriker skall finnas i dokumentet:

Hantering	*Hur kan projektet byggas och användas?*
Syfte	*Beskriv kortfattat vad som skall uppnås med arbetet.*
Genomförande	*Hur har arbetet genomförts? Vilka tillvägagångssätt, miljöer och verktyg har använts?*
Diskussion	*Reflektera över arbetets genomförande. Vad har varit bra, mindre bra, kunde något göras annorlunda eller bättre? Uppfyller lösningen arbetets syfte? Nämn även era slutsatser kring såväl projektarbetet som kursen i övrigt.*

Tänk på att presentera innehållet på ett snyggt och prydligt sätt. Använd MD-formattering för rubriker och placera eventuell kod i avsedda kodrutor!

För att era arbetsflöden skall kunna bedömas måste ni göra teamets **Trello Board** tillgängligt. Det är här valfritt om ni vill göra hela brädet *public* och då använda dess vanliga *URL*. Alternativt så kan brädet förbli *private* och en särskild länk genereras för att ge andra åtkomst. Detta görs i Trellomenyn för brädet;

`People → Invite → Invite people by giving them a special link...`

Slutligen skall varje gruppmedlem utföra en formell inlämning i avsedd inlämningslåda i Moodle och följande information skall läggas till som inlämningskommentar:

- *URL* till projektets repository på **Bitbucket**!
- *URL* till teamets **Trello board** (om privat; använd den genererade länken)!

OBS!! Inlämningen kräver att båda gruppmedlemmar klickar på ”skicka in” knappen i inlämningslådan!

4 Betygskriterier

Examinationen baseras på följande fyra delar;

Versionshantering *med Git och Bitbucket där alla projektmedlemmars bidrag tydligt framkommer utifrån versionshistoriken!*

Utvecklingsmetod; *hur väl ni förhåller er till…*

- ...Kanban som agil metodologi, med tillhörande fördelning av arbetsuppgifter (tasks) som är lämpligt avgränsade!
- ...upprättade krav gällande arbetsflödet (branchstrategi och granskningskontroll)!

Dokumentation; *hur väl lösningen uppfyller uppgiftskraven!*

Enhetstester; *hur väl lösningen uppfyller uppgiftskraven!*

Varje del bedöms utifrån ett fyragradigt poängsystem och för att bli godkänd på kursen krävs det att ingen del är underkänd. Betyget baseras sedan på den totala poängsumman från alla delar!

Bedömningspoäng	
Mycket bra	[3]
Bra	[2]
Mindre bra	[1]
Underkänd	[0]

Betygsskalor	
A	[11]
B	[9]
C	[7]
D	[6]
E	[5]
Fx	[4]

OBS!! Vid försenad inlämning sänks betyget ett steg!

Referenser

[1] Wikipedia, "Game of Life", 2017. [Online]
Tillgänglig: https://en.wikipedia.org/wiki/Conway's_Game_of_Life
[Åtkomst: 17 Oktober 2017]

[2] Atlassian, "Create and administer your team", 2017. [Online]
Tillgänglig: https://confluence.atlassian.com/bitbucket/create-and-administer-your-team-665225537.html
[Åtkomst: 17 Oktober 2017]

[3] Atlassian, "Comparing Workflows", 2017. [Online]
Tillgänglig: https://www.atlassian.com/git/tutorials/comparing-workflows
[Åtkomst: 17 Oktober 2017]

[4] Atlassian, "Making a Pull Request", 2017. [Online]
Tillgänglig: https://www.atlassian.com/git/tutorials/making-a-pull-request
[Åtkomst: 17 Oktober 2017]