Sensorrettleiing for INF112, våren 2022

Funksjonen til ei sensorrettleiing å være ei rettleiing til sensor. Den skal ikkje fungere som ein fasit eller eit pedagogisk verktøy for studentane. Rettleiinga skal vidare ikkje være til hinder for at sensor utviser godt fagleg skjønn. Sensorrettleiing er offentleg og skal være tilgjengeleg for studentane etter at eksamen er avvikla. Det er tilstrekkelig at den vert gjort tilgjengelig på førespurnad.

A. Formalia

- INF112 Innføring i systemutvikling

- Læringsutbyte for emnet:

Studenten vil vere i stand til å:

Studenten skal ved avslutta emne ha følgjande læringsutbyte definert i kunnskapar, ferdigheiter og generell kompetanse:

Kunnskapar

Studenten

- kan skildra standard programvareutviklingsprosessar,
- kan forklara viktige prinsipp for systemdesign,
- kjenner til dei viktigaste lover og avtaleverk for systemkonstruksjon (intellektuelle rettar, avtaleverk for arbeidsliv),
- kan teori for organisering av lagarbeid,
- kan forklare ein vanleg notasjon for analyse og design, og
- har kunnskap om fagområdet si historie, tradisjonar og plass i samfunnet.

Ferdigheiter

Studenten

- kan bruke etablerte teknikkar for å samle inn og analysere brukarkrav og behov,
- kan nytte fagleg kunnskap på praktiske problemstillingar og grunngje val,
- kan reflektere over eige fagleg skjønn og justere denne under rettleiing
- kan finne, vurdere og vise til informasjon og fagstoff og framstille dette slik at det klargjer ein problemstilling, og beherskar eit integrert utviklingsverkty (IDE) og eit versjonskontrollsystem.

Generell kompetanse

Studenten

- har innsikt i relevante fag- og yrkesetiske problemstillingar,
- kan planleggje og gjennomføre programvareprosjekt i team,
- kan planleggje og gjennomføre programvareprosjekt i tråd med etiske krav og retningslinjer,
- kan formidle sentralt fagstoff både skriftleg, munnleg og gjennom andre relevante uttrykksformer, og
- kan utveksle synspunkt og erfaringar med andre med bakgrunn innafor fagområdet og gjennom dette bidra til god praksis.

- Semester: Vår

- Antal studiepoeng: 10

- Vurderingsform og varigheit: Skriftlig skoleeksamen, 3 timer
- Emneskildring: https://www.uib.no/emne/INF112

B. Litteratur og undervising

Alle skrevne og trykte hjelpemidler var tilgjengelig på eksamen, men studentene var orientert om at medbrakt litteratur antakelig ville ha begrenset nytteverdi. En kort oppsummering av pensum + den ene pensumboken (om Kanban og Scrum) var vedlagt eksamensoppgaven som PDF.

Gruppearbeidet har utgjort den sentrale delen av undervisningen, både med tanke på trening av tekniske ferdigheter og erfaring med prosjektarbeid og teamdynamikk. Resultatet derfra utgjør 50% av karakteren (inkl. skriftlig og muntlig formidling av erfaringer fra prosjektarbeidet). Forelesningene har tatt opp tema innen kommunikasjon/teamarbeid/(enkel) psykologi, samt utviklingsmetodikk og -verktøy, og forskjellige tekniske tema. Studentene har ikke hatt seminarer med organiserte diskusjoner e.l.,

Se forøvrig kommentarer til vedlagt oppgavesett.

C. Andre relevante opplysningar

Følgende beskrivelse av eksamen var gjort tilgjenglig på forhånd. Se forøvrig kommentarer til vedlagt oppgavesett.

Om eksamen

- Vi kjører «åpen bok»-eksamen (dvs. alle skrevne og trykte hjelpemidler tillatt). Det vil si at spørsmålene i liten grad vil handle om konkrete fakta, men heller at du skal vurdere, reflektere over eller analysere noe.
- Tema for oppgavene vil falle innenfor:
  - Team-arbeid, kommunikasjon og sosiale ting
  - Utviklingsmetodikk, krav, planlegging, osv
  - Teknisk – abstraksjon, objekt-orientering, testing, design patterns, osv
  - Verktøy – versjonskontroll, bygging, testing, kvalitetssjekk, deployment
- Typisk form på oppgavene:
  - Forklar hvorfor / vurder i forhold til noe («hva er forskjellen på Scrum og Kanban? Hvilke fordeler/ulemper har de?»)
  - Gi eksempel på nytten av, eller riktig/feil bruk av noe («Hva er vitsen med *Factory* design pattern? Gi et eksempel som illustrerer dette»)
  - Gi tilbakemelding på noe – hva er «feil» i denne koden? («Hvordan vil det være å utvide/endre denne klassen til å også støtte var-zombier? Hvordan vil du evt. forbedre designet?»)
  - Tenk deg denne situasjonen (f.eks. team med kommunikasjonsproblemer), hva slags råd ville du gitt?
- For noen av oppgavene forventer vi korte svar, f.eks. noen setninger – dette står i oppgaveteksten

De viktigste tingene har du allerede lært når du jobbet med prosjektet! Bruk gjerne tid til å tenke gjennom erfaringene, og evt diskutere dem med andre.

Begrunnelser

Det er viktig å gi gode begrunnelser, f.eks.

- «I morgen skal dere ha møte med den nye sprint-eksperten på Scrum-teamet. Du har ansvar for å skaffe snacks, og Petter er kanin. Hva gjør du?»
    - ok svar: «Jeg vil gjerne at han skal føle seg velkommen og at vi alle får et positivt førsteinntrykk, så jeg kjøper inn gulrøtter, for jeg vet kaniner liker gulrøtter.» – Akkurat hva slags snacks du skaffer er ikke viktig, men du forklarer *hvorfor*, hva du prøver og oppnå og hvorfor du mener det er viktig for teamarbeidet.
    - dårlig svar: «Jeg kjøper gulrøtter.» – Kan være gulrøtter er den beste løsningen, men vi vil vite hva du har tenkt.

Valgoppgaver

På noen av oppgavene skal du velge én av deloppgavene og besvare *kun* denne, mens på andre skal du svare på alle deloppgavene – pass på å lese oppgaven nøye så du ikke gjør unødig arbeid.

Kilder

Vi har ikke sett så mye på formell litteratur om f.eks. møter og kommunikasjon, så det er ikke forventet at dere refererer til kilder på f.eks. hvordan man håndterer en uenighet i teamet.

Men:

- Hvis du henter informasjon fra skriftlige kilder du har med, er det naturlig at du refererer til kilden i besvarelsen.
- Hvis du siterer fra en kilde – f.eks. besvarer eksempelspørsmålet om Scrum og Kanban med å (blant annet) kopiere eller omskrive fra side 49/50 i Scrum/Kanban-boken, *må* du gjøre det tydelig hvor du har hentet ting fra og om det er et direkte sitat eller en omskriving – noe annet er fusk
- Dette gjelder selvfølgelig også om du kopierer eksempelkode. Tenk også på at hvis spørsmålet er «lag et eksempel» så imponerer du ikke sensor ved å bruke samme eksempel som i boken!

Les mer på UiBs sider om kilder

D. Vedlegg

- Oppgåvesett
- Pensum (ev. lenke til Leganto, emneskildring): https://bibsys-d.alma.exlibrisgroup.com/leganto/readinglist/lists/8590422980002207?institute=47BIBSYS_UBB&auth=SAML
- Karakterskala frå UHR, eventuelt lokale tilpassingar: https://www.uhr.no/_f/p1/i4bfb251a-5e7c-4e34-916b-85478c61a800/karaktersystemet_generelle_kvalitative_beskrivelser.pdf

Oppgave 1 (12%)

```java
public class Player extends Sprite {
    /* … */

    /** Update player position etc */
    public void update(float deltaTime) {
        if (Gdx.input.isKeyPressed(Keys.LEFT) && canGo(-1)) {
            deltaX = -speed;
        } else if(Gdx.input.isKeyPressed(Keys.RIGHT) && canGo(1)) {
            deltaX = speed;
        } else {
            deltaX = 0;
        }

        setX(getX() + deltaX);
    }
    /** Draw player with health bar */
    public void draw(Batch batch){
        super.draw(batch);
        drawHealthBar(batch);
    }

    /** Check if we can go in the given direction (-1 for left, 1 for right) */
    boolean canGo(int direction) { … }

    /* … */
}

```

Studer Player-klassen i PDF-dokumentet og **besvar de tre spørsmålene**:

*Oppgaven er ment å teste forståelse for seperation-of-concerns, MVC, designprinsipper; samt praktisk enhetstesting (enten med mocking eller ved å gjøre bedre designvalg). Problemer med enhetstesting ifm. brukerinteraksjon var et vanlig problem i prosjekten i løpet av semesteret.(**(!)** = ekstra viktig)*

a) Hvordan vil du gå frem for å teste update-metoden? Forklar kort.

- **(!)** Bør teste: at Player beveger seg til høyre/venstre eller står stille avhengig av input. At Player ikke beveger seg hvis det ikke er lov å gå i den aktuelle retningen.
- Kan gjøres med noen få scenarier, f.eks. alle kombinasjoner av LEFT, RIGHT og canGo, eller evt. ved å generere en strøm av tilfeldig input
- Metoden er tricky å teste fordi den *poller* tastaturet så det er ikke bare å sende inn tastetrykk som parameter. Redesign (f.eks. med eventhåndtering) vil kunne løse det problemet.
- **(!)** Mocking av isKeyPressed og canGo (+ evt. setX/getX) vil løse problemet
- Bruker coverage-verktøy for å se at man har dekket alle grenene i koden
- (Hva skjer/bør skje hvis både LEFT og RIGHT er trykket ned?)
- (Hvis man ikke mocker libgdx-funksjonaliteten må man antakelig ordne testoppsettet så libgdx starter i headless-modus)

b) Hvilke problemer ser du for deg om klassen skal brukes f.eks. på en mobiltelefon eller en spillkonsoll uten tastatur? Hva ville du gjort annerledes for å gjøre det enklere å støtte flere plattformer? Forklar kort.

- **(!)** update() sjekker spesifikke taster og vil derfor ikke virke på systemer uten tastatur; og input fra touchskjerm og spillkontroller lar seg ikke nødvendigvis oversette til enkle tastetrykk

- **(!)** inputhåndteringen burde vært skilt ut fra Player-klassen – f.eks. at Player implementerer et API for styring som er uavhengig av inputmetode (kanskje goLeft/goRight/stop, eller move(±factor) e.l.)

- evt. kan man styre ved hjelp av krefter/akselleråsjon i et fysikk(-aktig) system, samme API kan da brukes av andre spillelementer som skal bevege spilleren

- sammenveving av modell med grafikk og input kan også gi andre problemer når man skal støtte andre plattformer

- det er selvfølgelig mange forskjellige måter å håndtere variert input på, men vi vil nok ihvertfall ha lyst til å abstrahere vekk fra konkrete taster – ideelt sett bør ikke Player-klassen merke forskjell på om den blir styrt av et tastatur, en gamepad, en AI eller testcase e.l.

c) Hva tenker du generelt om designvalgene i kodesnutten? Er det noe annet du ville gjort forskjellig? Forklar kort.

- **(!)** Designet blander funskjonalitet som tilhører view (input og grafikk) med modellen (forretningsregler) – dette er gjerne en dårlig ide, og gjør koden vanskeligere å gjenbruke, teste og endre.

- => Det kan være lurt å benytte MVC eller et tilsvarende skille mellom modell, input og grafikk

- **(!)** Bruk av ±1 for å uttrykke retning kan føre til forvirring (er -1 venstre eller høyre? vil canGo(-2) sjekke om jeg kan gå *to* skritt til venstre, eller ser den bare på retningen?)

- Håndtering av canGo() og bevegelse generelt er gjerne noe som er felles med andre aktører i spillet, og kan kanskje flyttes til en superklasse eller hjelpeklasse. Tilsvarende med "tegn med healthBar".

- (Koden kan fort bli komplisert om man skal utvide med andre taster/bevegelsesmønstre)

- (Ellers er det mange småting det går an å klage på, men det går mer på stil enn på design: kommentarene er litt vage og/eller intetsigende (at det bare er én linje og ikke full JavaDoc kan kanskje være for å spare plass i eksamensoppgaven) – deltaTime er ikke i bruk (kanskje det er meningen update() skal @Override noe i Sprite-klassen?) – oppdatering av posisjon kan kanskje gjøres bedre (sette deltaX i en ny metode) eller enklere (dropp deltaX). Men: switch i stedet for if ville ikke funket i dette tilfellet.)

Oppgave 2 (12%)

Fordi du har gjort en så bra jobb i INF112 i vår, har Universitetet bedt deg om å være foreleser neste vår.
Tanken med oppgaven er at kandidaten skal reflektere over egen og andres læring, og vise innsikt i produkt- og prosjektplanlegging.

a) **Gi en kort beskrivelse** av et prosjekt du tror vil egne seg for INF112-studentene. *(2–3 setninger + forklar tankegangen din)*

- Et kort, konkret og avgrenset konsept – f.eks. «Lag et go-kart spill i 3D. Spillet skal støtte inntil fire spillere på samme skjerm»
- Noen forskjellige ting å tenke gjennom ved valg av oppgave:
  - Er studentene kjent med konseptet fra før, eller må de sette seg inn i problemdomenet? (Det kan være nyttig trening å bruke litt tid på å sette seg inn i bilspill eller prosjektplanleggingssystemer, men antakelig for tidkrevende å sette seg inn i f.eks. flysimulatorer eller kompilatorer)
  - Har de sjans til å oppnå noe innen rimelig tid, og bli ferdig i løpet av semesteret?
  - Er konseptet ekskluderende eller mindre motiverende for enkelte (grupper av) studenter? Går det evt. an å la studentene fylle konseptet med noe de er interessert i?
  - Trenger man spesialkompetanse for å gjennomføre det?
  - Egner prosjektet seg for å trene studentene i aktuelle læringsmål? (f.eks. Scrum-metodikk, objektorientering, testing, API-design, sikkerhet, brukerinteraksjon, personvern, deployment e.l. – man kommer ikke innom alt, men det er lurt å sørge for at man er innom en viss bredde av aktuelle teknikker/problemstillinger. F.eks. er «superraske matrisekalkulasjoner med konsoll-I/O» nyttig og utfordrende, men også noe som vil gå på bekostning av SOLID-prinsipper osv)
- Noen eksempler på gode (enkelt)poenger fra besvarelsene:
  - Quiz-app/spill: «Ved at det skal lages flere spørsmål, kan det gi en mulighet for å bli bedre kjent. Du viste kanskje ikke at Per var veldig flink til å dyrke grønnsaker og viste derfor at agurk vokser på en plante på samme måte som en tomat.»
  - Lage læringsapp for 6–12-åringer: «Grunnen til dette prosjektet er for å føle at de kan utvikle noe som er nyttig for andre [… Det setter] krav til studentene om at programmet skal være lett å forstå»
  - Pinball-spill: «I tillegg er logikken i spill som ble oppfunnet for lenge siden enkel nok til at det er gjennomførbart for studentene nesten uten hjelp. Dette er nyttig både for studentene sin følelse av måloppnåelse, men også for at de skal oppleve å bygge noe **fra bunnen av.**»

b) Hva tenker du vil være passende *minimum viable product* (MVP)? **Lag en liste av krav** for studentene. *(5–10 krav + forklar tankegangen din)*

Her er vi ute etter en konkret avgrensing, i tråd med tankegangen i a).

c) Studentene er litt usikre på hvordan de skal besvare første obligen. **Gi dem et eksempel** på en passende *user story* for ett av MVP-kravene. *(1–3 setninger + forklar tankegangen din)*

(«user story» == «brukerhistorie»)

En brukerhistorie som 1) passer med prosjektet, 2) har en troverdig bruker, 3) har en god motivasjon, 4) er kort og konsis, 5) kan lede frem mot konkrete akseptansekriterier og arbeidsoppgaver

*Dårlig: «As **a goat** I want **more bushes and shrubs** so that I can **ruminate about the quality of my code**.»*
*Bedre: «As a **casual player** I want to **automatically save my progress**, so I can **switch off the game when my break is over**»* (Vi kan lett tenke oss en person som er «casual player», og å kunne slå av når som helst er en rimelig motiviasjon, og «save progress automatically» er funksjonalitet vi kan teste)

Oppgave 3 (14%)

SOLID er et akronym for fem designprinsipper for å gjøre programvare mer forståelig, oversiktlig, fleksibel og vedlikeholdbar (bedre kvalitet). Bokstavene står for (fra Wikipedia):

- *The **S**ingle-responsibility principle: "There should never be more than one reason for a class to change."* – hver bit av koden (fra funksjon/metode til klasse og pakke) har kun ett ansvarsområde.
- *The **O**pen–closed principle: "Software entities … should be open for extension, but closed for modification."* – kode bør kunne utvides ved å legge til kode (extends/implements), ikke ved å endre eksisterende kode
- *The **L**iskov substitution principle: "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it."* – når du arver/impementerer et grensesnitt må oppførselen være konsistent med (minst «like bra» som) superklassen
- *The **I**nterface segregation principle: "Many client-specific interfaces are better than one general-purpose interface."* – unngå å gjøre klientklasser avhengig av metoder de ikke bruker
- *The **D**ependency inversion principle: "Depend upon abstractions, [not] concretions."* – abstraksjoner (høynivå-moduler) skal ikke være avhengig av konkrete detaljer (lavnivå-moduler) (bruk interface!)

Velg *ett* av prinsippene og lag et eksempel som illustrerer hvorfor det er nyttig/hvordan det brukes.

Oppgaven er ment å teste forståelse for det aktuelle prinsippet og evne til å sette det i en kontekst med vanlige designutfordringer.

- Er eksempelet relevant?
- Er prinsippet korrekt anvendt?
- Dekker eksempelet hele eller bare deler?
- Er det ok forklart, lett å forstå og ikke unødvendig komplisert?
- Er eksempelet originalt, eller er det typisk eksempel (fra forelesning/WIkipedia e.l.)

(Gode eksempler er noe selv lærebokforfattere sliter med, så det er ingen forventing om at kandidatene skal lage «perfekte» eksempler)

Oppgave 4 (12%)

Velg *ett* av disse scenariene, og forklar kort hva du ville gjort. Husk å begrunne svaret / forklare vurderingene dine.

Oppgaven er ment å teste vurdering av og refleksjon over utfordringer med teamarbeid.

Her legger vi vekt på at svaret er godt motivert, ut fra prinsipper for kommunikasjon/team/utviklingsarbeid som vi har vært gjennom i emnet, og/eller erfaringer kandidaten har gjort seg i løpet av prosjektarbeidet, og/eller hvordan kandidaten ville gått frem for å skaffe mer informasjon/finne ut hva som er best å gjøre. *(Vi legger ikke vekt på hva sensor ville gjort selv!)*

Alternativ 1: Du har blitt ansatt på et lite utviklingsteam i en bedrift. Ledelsen/eierne er mest opptatt av seilbåtene sine og har hverken kompetanse til eller interesse av å styre utviklingsarbeidet. De siste månedene har teamet begynt å dele seg i to deler som jobber på hver sin branch og nesten ikke snakker sammen. Begge fraksjonene jobber i praksis med å løse samme problemet, men de har gjort helt forskjellige valg (f.eks. «all koden kjører på serveren med JSP» vs. «all koden kjører i nettleseren med React JS»). Hvordan vil du gå frem for å få teamet til å snakke sammen, samles om én løsning og jobbe sammen?

eller

Alternativ 2: Konsulentselskapet du jobber i har gitt sommerjobb til fire informatikkstudenter og bedt deg ta ansvar for at de skal ha en spennende og utfordrende sommer så de får lyst til å søke jobb hos dere i fremtiden. Hva vil du gjøre? Hva tenker du er viktigst at de får erfaring med, med tanke på utviklings- og teamarbeid?

eller

Alternativ 3: Du og tre andre studenter jobber sammen på et prosjekt (INF112 eller INF219, f.eks.). Du har fått en litt forsinket start på semesteret, så de andre er allerede i gang med kodingen når kommer inn i bildet. Det viser seg at de andre på gruppen ikke har satt opp noen form for versjonskontroll, automatisk testing eller *continuous integration*, slik du hadde forventet. Hvordan vil du gå frem for å forbedre situasjonen? Både med tanke på det tekniske, og hvordan du vil forklare/overbevise de andre om at det er verd bryderiet?

Oppgave 5 (50%)

Dette punktet skal ikke besvares. Her vil poengsummen fra prosjektarbeidet dukke opp og telles med i totalkarakteren.