Eksamen vår 2020 og noen tips til løsning

Oppgave 1 (10%)

Minimum Viable Product (MVP) er et verktøy for å komme seg tidlig i produksjon og få reell feedback på systemet som lages, før systemet er ferdigstilt. Roborally-spillet du og teamet laget hadde mange krav med i MVP. Kravene slik de ble fremsatt på forelesning:

- ha et spillebrett
- vise en brikke
- kunne flytte en brikke
- spille fra ulike maskiner
- dele ut kort
- velge kort (5 av 9)
- flytte brikke utfra kort
- dele ut nye kort ved ny runde
- vise flere (i alle fall to) brikker på samme brett
- dele ut kort til hver robot
- flytte flere brikker samtidig
- flytte brikker utfra prioritet på kort
- flagg på brettet
- kunne registrere at en robot har vært innom et flagg
- håndtere konflikter i bevegelse riktig
- kunne legge igjen backup
- restarte fra backup v ødeleggelse
- går du i et hull, blir du ødelagt, mister et liv og begynner fra forrige backup
- går du av brettet, blir du ødelagt, mister et liv og begynner fra forrige backup
- blir du skutt i fillebiter (9 i skade) blir du ødelagt, mister et liv og begynner fra forrige backup
- vender en robot mot deg ved slutten av en fase blir du skutt og får en i skade
- får du skade får du mindre kort i henhold til skaden du har
- kan ikke gå gjennom vegger
- for mykje skade brenner fast programkort fra runde til runde

I etterkant mener jeg MVP burde vært mer begrenset før dere gikk «i produksjon». **Sett opp et forslag til hvilke krav som du mener er tilstrekkelig for MVP. Grunngi hvorfor akkurat disse er valgt, og hvorfor du har valgt vekk andre. Krav som har felles egenskaper kan grupperes slik at disse kan beskrives felles.** Ta utgangspunkt i kravene som ble fremsatt på forelesning, men legg til eller endre krav hvis du mener det er nødvendig.

Løsningsforslag:

- *ha et spillebrett*

- *vise brikke*
- *flytte brikke*
- *besøke flagg*
- *kunne registrere at en robot har vært innom et flagg*
- *vinne spill (fordi flagg er besøkt)*

Ingenting annet er strengt nødvendig for å «spille» en full runde og vinne spillet. Spillbarheten er ikke særlig god, men det er nok til å hente inn erfaringer fra brukere. Hovedpoenget er å bevege roboten og kunne avslutte spillet ved å vinne. Siden det å vinne skjer ved å besøke et flagg og registrere det, må dette inn i MVP. Det å dele ut programkort er fint, men ikke nødvendig for å begynne å hente inn erfaringer. Samtidig ser jeg mange studenter gjør vurderingen at kortene (og dermed runder og faser) er en essensiell del av spillet, så vi trenger ikke trekke for denne, det er en helt OK vurdering. Tilsvarende med alle brettelementer, at robotene dør når de går av brettet eller i hull, skade osv. (Ser at jeg har glemt å ta med brettelementer som samlebånd og sånt i kravlisten her, så helt OK at de ikke nevner dem.)

Vurdering er viktig. Er grunnene til å ta med flere eller færre krav veloverveide? Poengmessig deles dette i to, der minst halvparten av poengene handler om kvalitet på vurderingene. Klarer du å klassifisere noen typer krav (feks ulike elementer på brettet) som tilsvarende?

Oppgave (22%)

Gå gjennom kodeeksempelet under. Bruk god tid på å gå gjennom koden! Det finnes både små og store problemer med koden. Du trenger ikke vurdere kode som ikke er vist. Du kan klippe og lime koden inn i en editor om du ønsker, men siden du ikke har resten av koden vil dette ikke kompilere, og det er heller ikke meningen.

1. **Hva gjør koden?** Svar kort (6 poeng)
2. Vurder kvaliteten på koden. **Fortell kort hva slags utfordringar og forbedringspotensiale du ser. Forklar hva slags refaktoreringsmetoder (med referanse til koden) du ville brukt for å forbedre koden** (8 poeng)
3. **Beskriv hva slags tester du ville hatt for å kunna refaktorere på en trygg måte.** Du trenger ikke skrive fullstendige tester, men en kort beskrivelse av input, forventet output og hva testen skal avdekke. (8 poeng)

```
package eksamen;
import java.util.List;
public class PizzaMeny {
    private DBConnection dbConnection;
    public double bestill(List<Pizza> pizzaer, boolean student, boolean
ekstraOst) {
        // maks med de ovnene vi har
        if (pizzaer.size() > 6) {
            throw new RuntimeException("Så mange pizzaer klarer vi ikke
lage");
        }
        double pris = 0;
```

```
        for (Pizza pizza : pizzaer) {
            int pizzaPris = pizza.getPris();
            pizzaPris = pizzaPris + 10; //økonomisk margin
            pris += pizzaPris;
        }
        pris += 50 * pizzaer.size(); //bunn
        pris += 15 * pizzaer.size(); //saus
        if (ekstraOst) {
            pris += 10 * pizzaer.size();
        }
        if (pizzaer.size() == 6) {
            pris = 4 * (pris / 3);
        } else if (pizzaer.size() == 3 || pizzaer.size() == 4 ||
pizzaer.size() == 5) {
            pris = 2 * (pris / 3);
        }
        if (student) pris = pris * 0.8; //studentrabatt på 20%
        for (Pizza pizza : pizzaer) {
            String sql = String.format("insert into PIZZA_SOLGT (type,
pris) values (%s, %d)", pizza.getType(), pizza.getPris() + 10);
            dbConnection.updateDB(sql);
            dbConnection.updateDB("insert into ORDRE (antall, pris) values
(" + pizzaer.size() + ", " + pris + ")");
            List<Ingrediens> brukteIngredienser = pizza.hentIngredienser();
            sql = PizzaUtils.lagSql(brukteIngredienser);
            dbConnection.updateDB(sql);
        }
        return pris;
    }
}
```

Løsningsforslag
Hva gjør koden?

- *Koden sjekker at bestillingen er gyldig, i alle fall med tanke på antall pizzaer som kan bestilles.*
- *Deretter beregnes pris på pizzaene, før rabatt trekkes fra.*
- *Deretter lagres hvilke pizzaer som er solgt, hva hver ordre har kostet, samt at antall ingredienser som er brukt telles ned, i databasen.*

Refaktoreringer:

- *Trekke ut i konstant (alle magiske tall)*
- *Generalisere if-ene, evt også trekke ut i egen metode*
- *Vekk med kommentarer som ikke trengs*
- *Rename hovedmetode, den gjør mer enn bestille*
- *Trekke ut db-manipulering i egen metode*
- *Egen metode på feilhåndtering*
- *Rename student → erStudent*
- *Samle prisberegning for hver pizza*
- *Forenkling av koden utover dette er også mulig, kanskje til og med fjerne de to for-løkkene?*

Forslag til tester:

- *gyldig ordre kan leveres (<7 pizzaer)*
- *for stor ordre avvises (7 pizzaer)*
- *ville også hatt 0/ -1 pizza for validering, men dette kan vi ikke forvente at de vil vurdere*
- *1 pizza får forventet pris uten studentrabatt*
- *1 pizza får forventet pris med studentrabatt*
- *1 pizza får forventet pris med ekstra ost*
- *1 pizza får forventet pris med ekstra ost og studentrabatt*
- *3 pizzaer fører til rabatt*
- *4 pizzaer fører til rabatt*
- *5 pizzaer fører til rabatt*
- *6 pizzaer fører til enda mer rabatt*
- *3 pizzaer med studentrabatt*
- *3 pizzaer med studentrabatt og ekstra ost*
- *3 pizzaer med ekstra ost*
- *Dbresultat er riktig (evt resonnering rundt hvordan dette skal testes på annen måte eller skrive eksplisitt at dette ikke skal testes)*

Generelt:

Det er vanskelig å kreve at du skal få på plass alle testene eller alle refaktoreringene. Vi vurderer resonnementet deres og kvaliteten på forslagene. Du bør klare å nevne rename/konstanter/kommentarer/trekke ut i metode.

Oppgave 3 (8%)

Dette semesteret har dere både jobbet ved å møtes fysisk, men også remote etter universitetet stengte. **Med utgangspunkt i erfaringene du gjorde deg i prosjektet, nevn 3 positive effekter ved å jobbe remote og 3 positive ting med å møtes fysisk for å jobbe**.

Løsningsforslag:

Noen positive punkter ved å jobbe remote:

- *mer effektive møter fordi det er smertefullt å sitte i meningsløse samtaler på audio eller video*
- *viktig å vite hvem som gjør hva –> bedre oversikt over pågående saker*
- *kan ikke belage seg på kroppsspråk for å forstå hverandre, må bli mer tydelig i kommunikasjon.*
- *Får fred til å jobbe i eget tempo*
- *Lettere å jobbe sammen fordi begge har tilgang til sitt eget utstyr*
- *Utnytter tekniske verktøy bedre*

Noen negative punkter ved å jobbe remote (kanskje noe av dette dukker opp i positivt for fysisk oppmøte)

- *Vanskelig å kommunisere hvis båndbredden er for lav (både audio, video)*
- *Fare for å misforstå hverandre fordi man ikke får med seg at den andre ikke forstår*
- *Vanskeligere å holde oversikt over hvem som gjør hva/sårbart hvis ikke medlemmene bruker feks prosjekttavle på en god måte*
- *Vanskelig å diskutere design sammen fordi man ikke kan tegne på ark/tavle*
- *Må finne andre verktøy*
- *Vanskelig å bli kjent fordi man ikke treffes (vanskeligere å kommunisere bra)*

Positive punkter ved å møtes fysisk:

- *Lettere å se hvordan andre reagerer*
- *Lettere å bli kjent når man sitter ansikt til ansikt*
- *Design kan være lettere: tegne og diskutere*
- *Ikke forsinkelse i kommunikasjon*
- *Lettere å hjelpe hverandre*

Oppgave 4 (10%)

Etter nedstengningen av universitet har Zoom blitt brukt aktivt i undervisning til å holde forelesninger og løse gruppearbeid. Se for deg at du utvikler et tilsvarende system som skal legge til rette for forelesninger og gruppearbeid. **Lag 2 brukerhistorier med akseptansekriterier og arbeidsoppgaver som viser typisk bruk av tjenesten for en student eller foreleser. Skriv kort dersom annen informasjon er viktig, der det ikke passer inn i enten brukerhistorien, akseptansekriteriene eller arbeidsoppgavene.**

Løsningsforslag:

Vektlegge en god form på brukerhistoriene, feks «Som <rolle> ønsker jeg <funksjonalitet> slik at <behov oppfylles>. Konkrete, testbare akseptansekriterier. Fornuftige arbeidsoppgaver (modellere/designe/implementere backend/frontend). 5 p per brukerhistorie, 2 p brukerhistorie, 2 akseptansekriterie og 1 p for arbeidsoppgaver. Vurder særlig kvalitet på de to første.