


Titel:	Kurs:	 Malmö Högskola
Deluppgift 4: Sekvensdiagram och C++ kod	DA337A C++ med objektorienterad design och analys)	
Skapad av:	Enhet:	
Jonas Wahlfrid	Spel	
Granskad av:	Ämnesområde:	
	Datavetenskap	
Godkänd av:	Version:	
	Issue 1 Draft C	
Filnamn:	Datum:	
Uppgift3SekvensdiagramIssue1DraftC.doc	2015-11-04	

Dokumenthistoria

Version:	Datum:	Skapad av:	Granskad av:	Godkänd av:	Kommentar:
Issue 1 Draft A	2015-10-24	Jonas Wahlfrid			
Issue 1 Draft B	2015-11-01	Jonas Wahlfrid			Minskat omfattning byte till enbart Together
Issue 1 Draft C	2015-11-04	Jonas Wahlfrid			Förtydligande i rött

Innehållsförteckning

Dokumenthistoria	1
1. Inledning.....	2
2. Mål	2
3. Förberedelser.....	2
4. Modelleringsmetodik	2
5. Analys & Design	2
6. Namngivningskonventioner	2
7. Deluppgiften.....	2
8. Godkännande.....	3
9. Bilaga 1 Beskrivning modell element	4
9.1. Use Case diagram.....	4
9.2. Activity diagram Play four in a row in real life without a computer	4
9.3. Class diagrams.....	4
9.4. Sequence diagrams	4
9.4.1. 01 Start application.....	5
9.4.2. 02 Start new game (enter settings)	5
9.4.3. 03 Play game	5
9.4.4. 04 HumanPlayer makes a move	5
9.4.5. 05 ComputerPlayer makes a move.....	6
9.4.6. 06 RemotePlayer makes a move	6
9.4.7. 07 Quit game	6
9.4.8. 08 Restart game.....	6
9.4.9. 09 About.....	6
9.4.10. 10 How to play (Help).....	6
9.4.11. 11 Exit (Exit application)	6
9.5. Object diagram Snapshot of a ongoing game.....	6
9.6. Component and deployment diagram for four in a row game	7
9.7. Beskrivning av paketen.....	7
10. Bilaga 2 Tids planering och granskningslista	8
10.1. Mötestillfälle vecka 45.....	8
10.2. Mötestillfälle vecka 46.....	8
10.3. Mötestillfälle vecka 47.....	8
10.4. Mötestillfälle vecka 48.....	8
10.5. Mötestillfälle vecka 49.....	8

1. Inledning

Denna deluppgift är den fjärde och sista som har som mål att skapa en design modell av ett fyra i rad spel. Uppgiften löses av en grupp bestående av två studenter.

2. Mål

Målet med uppgiften är att ge övning i objekt orienterad design.

3. Förberedelser

Innan denna uppgift påbörjas:

1. Läs igenom detta dokument.
2. Gör klart deluppgift 1 och 3 samt få ett godkännande på uppgifterna.
3. Repetera avsnitt 2 i kompendiet "OOAD – Objektorienterad analys och design".

4. Modelleringsmetodik

Studenterna ska använda principerna bakom registerkorts metodiken som beskrivs i "OOAD – Objektorienterad analys och design". Genom att studera studenternas version av "Four in a row game Use Case Model Survey" och "Four in a row game Supplementary specification" går det att hitta objekt, klasser, attribut, metoder, relationer och scenarier.

5. Analys & Design

Skillnaden mellan analys & design är flytande. Studenterna bestämmer själv när dom vill gå vidare från analys till design. Det går att spara analysen innan designen men detta krävs inte i denna uppgift. I design modellen ska alla meddelanden i interaktionsdiagrammen vara knutna till metoder. Designmodellen ska visa metoder med dess parametrar och returtyp, studenterna behöver inte implementera metoderna.

6. Namngivningskonventioner

All dokumentering i C++ koden ska ske på engelska. Klassernas ansvarsområde från CRC korten ska dokumenteras i h filerna i form av kommentarer. Metoderna ska kommenteras i h filerna enligt <https://docs.unrealengine.com/latest/INT/Programming/Development/CodingStandard/index.html#exampleformatting>

Metoder som kastar exceptions ska dokumentera detta med Javadoc.

7. Deluppgiften

Målet med deluppgiften är främst att skapa sekvensdiagram och C++ kod, men även andra diagram som beskrivs under rubriken "Bilaga 1 Beskrivning modell element" i detta dokument. UML verktyget Borland Together (Free 30 day trial per student) ska användas. En fördel med Borland Together är att det kan generera sekvensdiagram från C++ kod och även generera kod från sekvensdiagram. Visual Paradigms sekvensdiagram saknar helt koppling till kod, vilket gör verktyget mindre lämpligt. Följ installationsinstruktionerna i dokumentet "Komma igång med Together" för att komma igång med Together.

C++ koden ska innehålla attribut, metoder med parametrar och retur typer. Koden ska vara stubbad så att den kompilerar, men metoderna behöver inte implementeras fullt ut. Eftersom anropen ska visas krävs det även cpp filer. Även användningen dvs. anropet av en metod/konstruktor med argument ska framgå i koden.

Kravet på VG är förutom alla andra krav för godkänt:

1. Trådar och mekanismer för att stanna och sen väcka en tråd ska framgå enligt C++ standard bibliotek.
2. Flertråds programmering i C++ ska kunna förklaras.
3. Velfungerande modell.
4. Vældokumenterad kod.

8. Godkännande

Studenterna är godkända när handledaren gått igenom och godkänt alla punkter i ” Bilaga 2 Tids planering och granskningslista” i detta dokument. Innan granskning ska studenterna förvissa sig om att det som ska redovisas uppfyller kraven. Studenterna ska redovisa delar av uppgiften enligt tidsplanen i bilagan ”Bilaga 2 Tids planering” i detta dokument.

Koden till modellen ska gå att kompilera. Vidare krävs att kraven i detta dokument och dess bilagor är uppfyllda. Den enskilde studenten ska kunna redogöra för alla detaljer i modellen.

Modellen ska uppfylla kraven på Fyra i rad spelet som är dokumenterade i ”Four in a row game Use Case Model Survey” och ”Four in a row game Supplementary specification”.

Den enskilde studenten ska kunna redogöra för alla detaljer. När uppgiften är godkänd av handledaren ska koden zippas tillsammans med UML diagrammen i SVG format på its learning. Höger klicka på ett diagram och välj export > Modeling > Image för att spara ett diagram i Scalable Vector Graphics (SVG) format.

9. Bilaga 1 Beskrivning modell element

Nedan beskrivs vad modellen ska innehålla och vad som redan finns.

9.1. Use Case diagram

Use Case diagram for the four in a row game finns redan i "Four in a row game Use Case Model Survey" dokumentet. Användningsfallsdiagrammet ska inte skapas i UML verktyget.

9.2. Activity diagram Play four in a row in real life without a computer

I "Four in a row game Use Case Model Survey" finns det ett aktivitetsdiagram som visar när två personer spelar fyra i rad utan dator stöd. Aktivitetsdiagrammet visar hur deltagarna spelar ett fyra i rad spel på ett spel med fysiska brickor och en ram samt mänskliga spelare. Eftersom detta är vår verksamhet kan vi säga att har utfört en verksamhetsmodellering. Aktivitetsdiagrammet ska inte skapas i UML verktyget.



9.3. Class diagrams

Klasser från samtliga paket ska visas på klassdiagrammet "All Classes". Alla associationer ska vara namngivna eller försedda med rollnamn. Multipliciteten (multiplicities) ska anges på samtliga associationer. Om det bara går att navigera i en riktning ska det framgå av pilen på associationen. Synligheten ska vara privat på attribut och metoder som inte behöver synas utåt. All C++ kod som genereras/implementeras ska kunna kompileras. Detta innebär att stubb kod måste skrivas. Klasser som kastas vid undantag ska visas på klassdiagrammet samt sekvensdiagram. Se <http://www.cprogramming.com/tutorial/exceptions.html> för en beskrivning av C++ undantags hantering. Metoder som kastar exceptions ska dokumentera detta med Javadoc. Tabellen nedan visa förslag på undantag.

Exceptions:
The BoardFullException is thrown when the board(game area) is full and no more moves can be placed.
The NotValidMoveException is thrown when a move is not valid. Ex a move outside the game area or when the column is full.
The WeHaveAWinnerException is thrown when a player has made a winning move.
The MoveRequestedException is thrown when the other player requested a move.
NetworkException is thrown when the network fails.

9.4. Sequence diagrams

Sekvensdiagram används för att visa hur ett användningsfall kan realiseras. På sekvensdiagrammet visas ett antal objekt som utbyter meddelanden (metod anrop). Objekt behöver klasser för att kunna instansieras och kräver därför att det finns klasser. Under modellering hittas ibland klassen först och andra gånger objektet först. För att ett objekt ska kunna anropa ett annat, krävs att det finns en länk till det andra objektet. Länken instansieras av en association som implementeras med en pekare/referenser i C++. För att ett objekt ska kunna erbjuda en metod måste det finnas metoder definierade i klassen. Ett sätt att hitta metoder är att gå igenom ett användningsfall och säkerställa att det realiseras på ett sekvensdiagram.

I Together går det som ett första steget skriva ett meddelande i form av en text t.ex. "add" från ett objektet aController till objektet aCalculator. I steg två kopplas objekten till sina klasser. I steg tre skapas en metod med parametrar och retur typ `IntegerNumber* Calculator::add(IntegerNumber* integerA, IntegerNumber* integerB)`. I steg fyra genereras kod i det anropande objektet med argument `integerAnswer = aCalculator->add(integerA, integerB);` (höger klicka för att generera kod). Ett problem är att Together genererar implementerings kod i h filen, detta går att komma runt genom att samtidigt skriva metoden i en editor, och sen välja metoden i sekvensdiagrammet. Ett värre problem är att det är lite omständligt att skapa sekvensdiagrammet.

Ett alternativ är att först skissa på papper, Together text meddelanden, ritprogram, löpande text eller

enbart i huvudet. Skissen används som indata för att skriva stubbkod. När ett avsnitt stubbkod är skriven används Together för att generera ett sekvensdiagram, höger klicka på en metod och välj "Generate sequence diagram". Sekvensdiagrammet används för att visualisera hur användnings fallet realiserar, så att lösningen kan verifieras.

Studenterna väljer själva hur dom bäst kommer fram till att bra designad/genomtänkt stubbkod skrivs. Förutom att C++ koden ska kompilera ska den kunna användas för att generera sekvensdiagram. **Under nedanstående rubriker beskrivs olika sekvensdiagrammen till olika användningsfall, det är tillåtet att generera ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen.**

9.4.1. 01 Start application

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Detta sekvensdiagram ska beskriva use case "Start application" och visa vad som händer när applikationen startas. I dokumentet "01StartApplicationSequence" finns ett förslag på hur detta diagram kan se ut med Java som mål språk. I "01StartApplicationSequence" dokumentet är det även visat hur en lösning, som möjliggör att Controller känner till MainWindow men MainWindow känner inte till Controller. Detta är löst genom att låta Controller göra ett anropa i MainWindow när Controller väntar på kommando från användaren. I detta anrop kör main-tråden wait() för att vänta på ett kommando. När operatören sedan klickar på en meny anropas notify() av AWT-Event tråden så att main-tråden återgår till att exekvera. (main-tråden och AWT-Event tråden skapas automatiskt, så det är inget som behöver modelleras. I "01StartApplicationSequence" dokumentet visas även vad som händer när Controller tagit emot ett kommando. Då tolkas detta och beroende på kommandot görs olika anrop.

9.4.2. 02 Start new game (enter settings)

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Detta sekvensdiagram ska visa use case "Start new game". Här visas hur ett spel startas och vilka objekt som skapas för att kunna spela spelet. Det ska framgå utav diagrammet att olika spelare skapas beroende på vilken spel typ som är vald t.ex. HumanPlayer vs. ComputerPalyer. I detta diagram bör det även framgå hur exekveringen delas i en ny tråd som sköter själva spelet. Observera att det detta diagram visar är en detaljering av det anrop som görs i diagrammet "01 Start application". I dokumentet "02StartNewGame" finns ett förslag på hur detta diagram kan se ut med Java som mål språk.

9.4.3. 03 Play game

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Detta sekvensdiagram ska visa use case "Play game". I detta diagram visas hur spelet spelas genom att visa vilka objekt som anropar varandra. I detta diagram ska det inte framgå hur de olika spelarna gör sina drag, däremot att dom ska göra ett drag. Detta kan visas genom att anropa klassen Player då drag ska göras. Hur spelarna gör sina drag visas istället i diagrammen "HumanPlayer makes a move", "ComputerPlayer makes a move" och "RemotePlayer makes a move". Genom att göra på detta sätt kan spelandet göras generellt och visas i ett diagram istället för ett för varje spel typ. Även undantag ska visas i sekvensdiagrammen.

9.4.4. 04 HumanPlayer makes a move

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Detta sekvensdiagram ska visa hur ett objekt av klassen HumanPlayer gör sitt drag. Diagrammet behöver

endast visa vad som händer i metoden för att göra ett drag. Hur metoden anropas visas i diagrammet "03 Play game".

9.4.5. 05 ComputerPlayer makes a move

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Detta sekvensdiagram ska visa hur ett objekt av klassen ComputerPlayer gör sitt drag. Diagrammet behöver endast visa vad som händer i metoden för att göra ett drag. Hur metoden anropas visas i diagrammet "03 Play game".

9.4.6. 06 RemotePlayer makes a move

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara en bra för läsbarheten.

Detta sekvensdiagram ska visa hur ett objekt av klassen RemotePlayer gör sitt drag. Diagrammet behöver endast visa vad som händer i metoden för att göra ett drag. Hur metoden anropas visas i diagrammet "03 Play game". I dokumentet "06RemotePlayerMakesAMoveSequence" finns ett förslag på hur detta diagram kan se ut med Java som mål språk.

9.4.7. 07 Quit game

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Här visas hur spelet avslutas. Ett exempel finns i dokumentet "Quit game sequence". I dokumentet "07QuitGameSequence" finns ett förslag på hur detta diagram kan se ut med Java som mål språk.

9.4.8. 08 Restart game

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Här visas hur spelet startas om. I dokumentet "08RestartGameSequence" finns ett förslag på hur detta diagram kan se ut med Java som mål språk.

9.4.9. 09 About

Detta sekvensdiagram behöver inte skapas, det är för litet för att tillföra något i övningssyfte, tanken var att visa use case "About".

9.4.10. 10 How to play (Help)

Detta sekvensdiagram behöver inte skapas, det är för litet för att tillföra något i övningssyfte, tanken var att visa use case "How to play".

9.4.11. 11 Exit (Exit application)

Det är tillåtet att generera totalt ett eller flera sekvensdiagram som visar på realiseringen av användningsfallen. Ni behöver inte dela ner sekvensdiagrammen i mindre avsnitt, men det kan vara bra för läsbarheten.

Detta sekvensdiagram visar use case "Exit (Exit application)". Vad som händer då applikationen avslutas. I dokumentet "11ExitSequence" finns ett förslag på hur detta diagram kan se ut med Java som mål språk.

9.5. Object diagram Snapshot of a ongoing game

Ett pågående spel ska exemplifieras med ett objektdiagram.

9.6. *Component and deployment diagram for four in a row game*

De olika paketen i din designmodell betraktas som komponenter. Rita ett komponentdiagram som visar beroendet mellan komponenterna. Det går bra att kombinera ett komponentdiagram och ett realiseringsdiagram till ett diagram. Ett realiseringsdiagram ritas, två datorer och en nätverkskoppling.

9.7. *Beskrivning av paketen*

Klasserna är strukturerade i följande paket:

- Paketet "general" här ska klasserna som realiserar spelets huvudfunktion sparas.
- Paketet "gui" här sparas klasser som realiserar användargränssnittet.
- Paketet "network" här sparas klasser som realiserar kommunikationen med ett annat spel.

Klasser från samtliga paket ska visas på klassdiagrammet "All Classes".

10. Bilaga 2 Tids planering och granskningslista

Vid varje mötes tillfälle ska det finnas ett klassdiagram genererat från koden som innehåller alla klasser inklusive undantagen. Associationerna ska vara namngivna eller roller angivna, multiplicitet angiven, riktning, inkapsling med privat på attribut och privata metoder. Användningsfallen för det aktuella tillfället ska finnas realiserade i kod och sekvensdiagram. Koden ska kunna kompileras och sekvensdiagrammet genereras. Samtliga granskningspunkter till respektive mötestillfälle ska vara klara.

10.1. Mötestillfälle vecka 45

- ☐ Together ska vara installerat och provat på heltalsräknaren enligt dokumentet "Komma igång med Together".

10.2. Mötestillfälle vecka 46

- ☐ **Class diagram:** Alla klasser inklusive undantag ☐ Namn på associationer eller roller
☐ Multiplicitet ☐ Riktning ☐ Inkapsling
- ☐ **Sequence:** 01 Start application ☐ Kompilerar, generera sekvensdiagram

10.3. Mötestillfälle vecka 47

- ☐ **Class diagram:** Alla klasser inklusive undantag ☐ Namn på associationer eller roller
☐ Multiplicitet ☐ Riktning ☐ Inkapsling
- ☐ **Sequence:** 02 Start new game (enter settings) ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc

10.4. Mötestillfälle vecka 48

- ☐ **Class diagram:** Alla klasser inklusive undantag ☐ Namn på associationer eller roller
☐ Multiplicitet ☐ Riktning ☐ Inkapsling
- ☐ **Sequence:** 03 Play game ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc
- ☐ **Sequence:** 04 HumanPlayer makes a move ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc
- ☐ **Sequence:** 05 ComputerPlayer makes a move ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc
- ☐ **Sequence:** 06 RemotePlayer makes a move ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc
- ☐ **Sequence:** 07 Quit game ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc
- ☐ **Sequence:** 08 Restart game ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc
- ☐ **Sequence:** 11 Exit ☐ Kompilerar, generera sekvensdiagram ☐ Javadoc

10.5. Mötestillfälle vecka 49

- ☐ **Object diagram:** Snapshot of an ongoing game
- ☐ **Component & Realisation diagram:** Component and deployment diagram for four in a row game