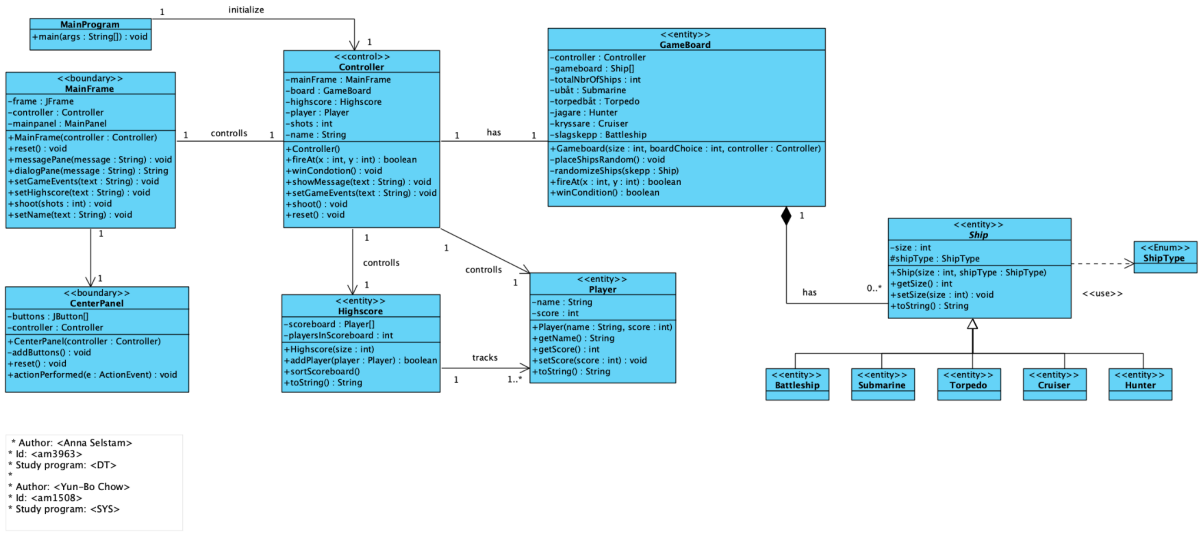
GRUPP 2
Kurs: DA339A

Anna Selstam
950616-5480
ID: am3963 (DT)

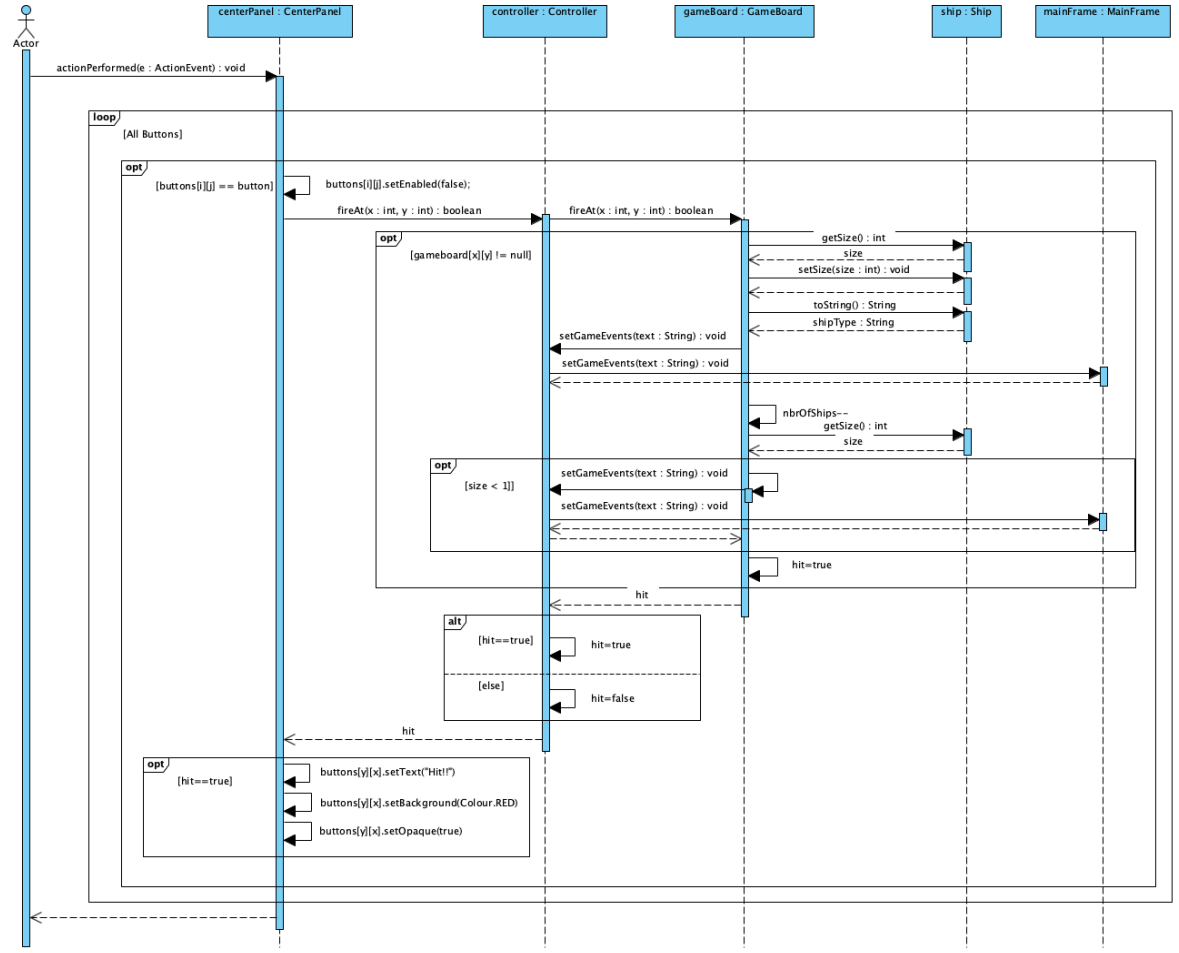
Yun-Bo Chow
021113-9035
ID: am1508 (SYS)

KLASSDIAGRAM



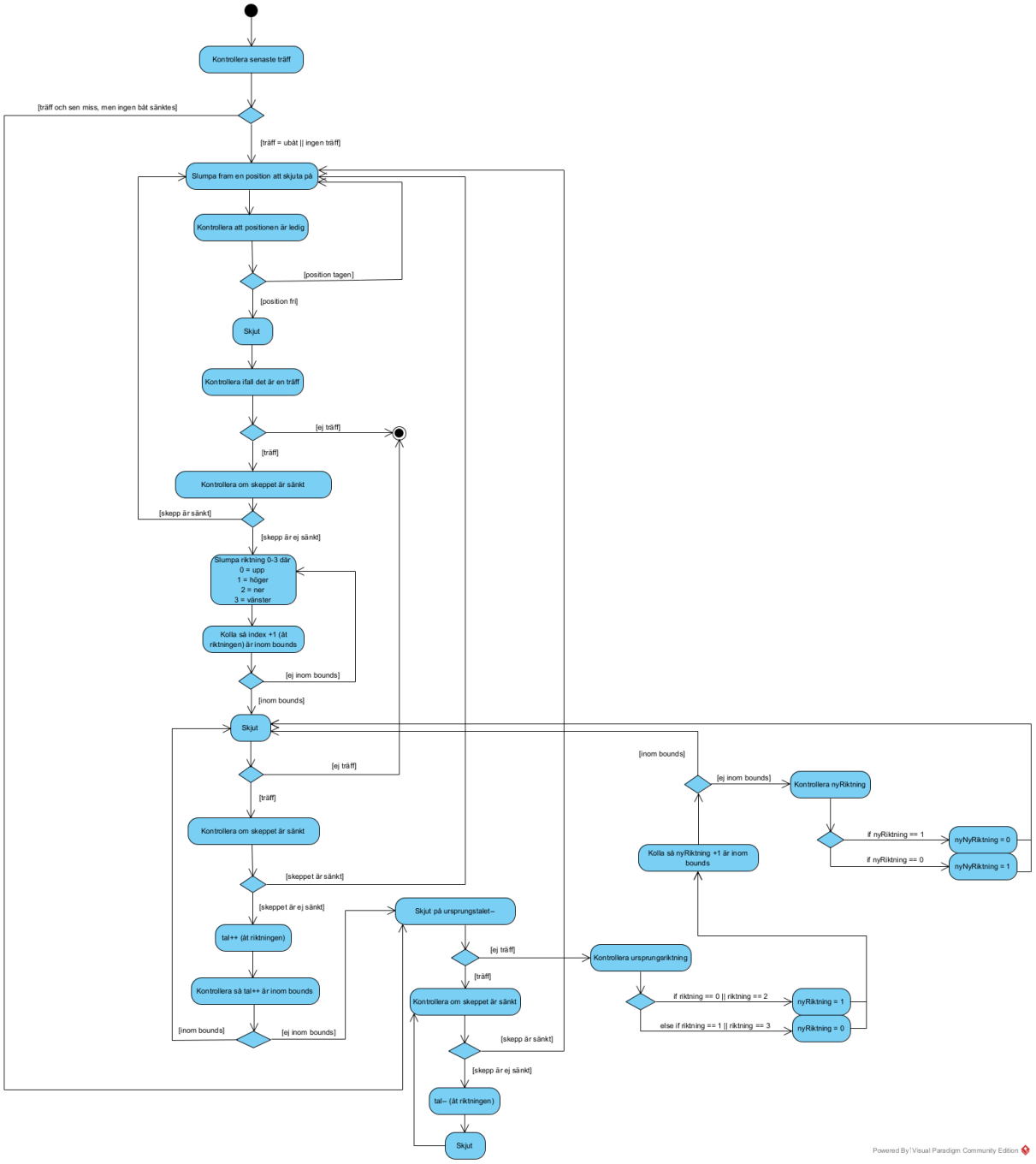

```
* Author: <Anna Selstam>
* Id: <am3963>
* Study program: <DT>
*
* Author: <Yun-Bo Chow>
* Id: <am1508>
* Study program: <SYS>
```

SEKVENSDIAGRAM



* Author: <Anna Selstam>
* Id: <am3963>
* Study program: <DT>
*
* Author: <Yun-Bo Chow>
* Id: <am1508>
* Study program: <SYS>

AKTIVITETSDIAGRAM



IU4TVG1 Svar: Presenterad algoritm hade gett ett bättre resultat då AI:n baserar sina skott på tidigare kunskap om ifall en bit av ett skepp redan är träffat eller ej. I detta program testar AI:n kringliggande platser om sin träffade position och fortsätter i det spåret tills det att skeppet är sänkt varpå den sedan slumpar en helt ny ruta.

Denna algoritm kan tillämpas för en duell mellan en människa och AI utan några förändringar.

IU4TG1 Svar:
Koden för olika typer av skepp generaliseras genom att ha en superklass vid namn Ship som alla subklasser ärver. Subklasserna är de specifika skeppen. Superklassen innehåller inga abstrakta metoder då koden strukturerades så att subklasserna inte innehåller attribut eller metoder. Attributerna i superklassen 'Ship', instansieras genom subklassens konstruktormetod som i sin tur anropar superklassens konstruktormetod med värdena som argument. Nackdelen med detta är att det kan tyckas vara en olämplig generalisering då subklasserna inte innehåller unika/specifika attribut. Däremot skapar denna struktur en smidigare hantering av spelplanen, och lättare förståelse för koden. Dessutom möjliggör detta metoder såsom 'instanceof' och 'equals' vid hanteringen av en träff samt slumpmässig placering av skepp då man behandlar skeppen som olika objekt snarare än olika instanser av samma objekt. Därav är generaliseringen lämplig och utnyttjas bra då de olika skeppen är lika, fast med några detaljer som skiljer dem åt. Utöver det har skeppen samma beteende, därav behöver inga metoder överskuggas i subklasserna.

Interface implementerades inte i lösningen då inga metoder behöver överskuggas inom subklasserna samt det faktum att koden strukturerades så att inga specifika metoder behöver existera inom subklasserna. Detta då de olika skeppen besitter samma beteende medan det som skiljer åt är skeppens värden på attributerna.

IU4TG2 Svar:
En del klasser/kod/objekt hade behövts dubbleras. Vid initialisering hade det behövts två stycken GameBoards med varsin uppsättning skepp, samt två stycken Players som instansieras av Highscore-klassen. Önskas det att skeppen placeras ut enligt spelarna själva får man ersätta den nuvarande algoritmen med detta, men annars får man anropa randomiseringsmetoden två gånger - en för varje spelare. Koden hade inte behövts förändras särskilt mycket förutom dubbleringen och ingen ny klass hade behövts förutom 'LeftPanel' i view (boundary) där den andra spelplanen skapats. Detta hade såklart inte varit nödvändigt men det följer vår nuvarande struktur. Även tillägg i Controller hade behövts som har koll på vems tur det är och som hanterar algoritmen gällande detta, samt en del villkorsändringar i funktioner. Detta hade inneburit den största förändringen. Spelet hade avslutats när en av spelarna når winCondition(), varpå vinnarens statistik hade registrerats i highscore listan.

Frågor till andra gruppen (IU4TG3)

Fråga 1:
Hur gjorde ni med randomiseringen? Och om ni inte har gjort den, hur tror ni att ni hade gjort?

Fråga 2:
Hur kopplade ni GUI:ts spelplan med spelplanen med skepp?

Fråga 3:
Hur kollade ni om platsen man sköt på var kopplad till ett skepp?

Våra svar…

Fråga 1: Hur gjorde ni med randomiseringen?
Svar: Först börjar vi med att slumpa en 1a eller 0a som bestämmer om skeppet ska vara vertikalt eller horisontalt och slumpar även ett tal inom storleken av spelplanen. Vi adderar sedan skeppets storlek till det talet och kollar ifall det är inom bounds. Är det inte inom bounds betyder det att skeppet ligger längs gränsen och vi vänder istället skeppet åt andra hållet och går vidare.

Är det inom bounds, går vi vidare och kollar om platsen är fri genom att använda oss av variabeln som avgör om skeppet ligger vertikalt eller horisontellt, lägger på skeppets storlek och kollar om det finns något 'instanceOf' inom de värdena. Är platsen inte fri börjar vi om från början och försöker igen. Om allt stämmer, placerar vi ut skeppet på brädet.

Fråga 2: Hur kopplade ni GUI:ts spelplan med spelplanen med skepp?
Svar: Vi har två delar: en 2D array i GUI:t som motsvarar knapparna och en 2D array som motsvarar Spelplanen. Spelplanen består av 10 x 10 'Ship' dvs vår Superklass. Inom denna spelplan randomiseras skeppens positioner och när man trycker på en knapp i GUI:t, skickar en ActionListener information till controller via funktionen 'fireAt' där koordinaterna skickas in och jämförs med spelplanens motsvarande koordinater.

Eventuell Fråga 3: Hur kollade ni om platsen man sköt på var kopplad till ett skepp?
Svar: x och y koordinaterna kommer in till metoden via GUI:t. Vi börjar med att kolla ifall den platsen är null eller inte. Är den null returnerar vi false och GUI:t sätter knappen som intryckt och ej tillgänglig. Är knappen inte null minskar vi spelplanen med 1 eftersom vi får en träff och skriver ut vilken typ av skepp man träffade genom att kalla på toString metoden för skeppet som ligger på de koordinaterna. Vi minskar även numret som utgör 'det totala antalet rutor som är skepp' med 1 för att detta numret sedan är avgörande om när man vinner. Till sist markerar GUI:t knappen som "Hit!" med rödmarkerad färg och gör den otillgänglig.