Projektbeskrivning

<Drakborgen>

2019-03-08

Projektmedlemmar:
Viktor Blidh <vikbl327@student.liu.se>

Handledare:
Mariusz Wzorek <mariusz.wzorek@liu.se>

Table of Contents

Planering

1. Introduktion till projektet

Under det projektet ska ett brädspel vid namn Drakborgen utvecklas. Detta brädspel lanserades 1985 av spelföretagen Alga och Brio och syftet med det här projektet är att skapa en digital kopia av det spelet. Drakborgen spelas med upp till fyra spelare och varje spelare väljer i början av spelet en hjälte. Varje hjälte har fyra olika egenskaper. Dessa är styrka, rustning, tur samt kroppspoäng och representeras av en siffra. Egenskaperna kommer påverka hur interaktioner med fällor och monster går. Målet är att ta hjälten från en av fyra startpunkter i hörnen till skattkammaren i mitten av brädet, sno så mycket skatter som möjligt, och sedan ta sig ut (komma tillbaka till en startpunkt) innan tiden tar slut.

2.Ytterligare bakgrundsinformation

Drakborgen är ett turordningsbaserat brädspel. Spelare turas om att utföra någon handling under sin tur och när alla spelare har genomfört sin runda räknas en timer ned. Denna timer indikerar hur många rundor som återstår innan alla spelare som inte tagit sig ut ur spelplanen förlorar.

Varje runda erbjuder ett antal olika möjliga handlingar för en spelare att utföra. Den vanligaste handlingen är att försöka ta sig till en ny ruta på spelbrädet. Spelaren väljer då åt vilket håll hen vill gå (möjliga håll bestäms av brickan hen står på) och drar därefter en rumsbricka. Denna bricka bestämmer hur rummet som spelaren går in i ser ut. Det kan vara en återvändsgränd, en T-korsning, en fälla med mera. När spelaren gått in rummet ska hen i de flesta fall dra ett rumskort. Rumskortet innehåller information om vad som finns i rummet. Oftast visar rumskortet ett tomt rum, men det kan också finnas monster, smycken, eller fällor. Om en möter ett monster initieras ett stridsläge, mer om det finns under rubriken Stridsläge. Om rumskortet visar en fälla kommer ett tal slumpas och jämföras med någon av hjältens egenskaper, oftast rustning eller tur. Vilken egenskap beror på typen av fälla som spelaren stött på.

Om spelaren inte vill dra en rumsbricka kan hen istället välja att dra ett rumsletningskort. Detta kan vara en bra idé om rummet en befinner sig i är en återvändsgränd eller har öppningar åt fel håll. När en spelare använder rumsletning kan denne finna en lönndörr och därefter få fortsätta rundan med att dra en rumsbricka och lägga den åt det håll som lönndörren pekade. Istället för en lönndörr går det också att finna smycken, fällor eller ingenting alls. I dessa fall behandlas det som dykt upp och därefter är rundan över.

Stridsläge

I drakborgen finns det fem olika monster. Dessa monster är:

- Svartalv
- Bergstroll
- Skelett
- Orch
- Två orcher

När en spelare stöter på ett monster ska denne besluta huruvida hen vill attackera, avvakta eller fly. Om spelaren attackerar eller avvaktar finns chansen att monstret flyr, annars kommer det att gå till attack. Om spelaren istället väljer att fly kan monstret lyckas hugga spelaren i ryggen innan denne lyckas fly, eller så misslyckas flykten helt och det blir strid ändå. Om flykten lyckas flyttar spelaren sin hjälte tillbaka till rumsbrickan hen stod på rundan innan. Om det blir en strid får spelaren välja mellan tre olika typer av attacker: A, B och C. Monstret har samma typer av attacker som väljs slumpvis. När spelaren valt attack jämförs den med monstrets attack. Reglerna som gäller är: A slår B, B slår C och C slår A. Förloraren tappar ett kroppspoäng. Monstrets kroppspoäng är okänt för spelaren och slumpas vid starten av striden. Striden är över när antingen monstrets eller hjältens kroppspoäng är noll.

Skattkammaren

När en spelare når den två brickor stora skattkammaren i mitten av brädet får denne i samma drag dra två skattkammarkort. Korten innehåller skatter och kan vara allt mellan mynt och jätterubiner. Skattkammaren vaktas dock av en sovande drake. Efter att spelaren tagit två skatter måste hen kolla om draken vaknar eller inte. Första gången är chansen att draken vaknar en på åtta. Om spelaren väljer att stanna kvar i skattkammaren och dra nya skattkammarkort nästa runda kommer chansen vara en på sju och så vidare. Om flera spelare står i skattkammaren samtidigt kommer chansen öka för varje gång någon dra skattkammarkort. Så fort skattkammaren är tom återställs riskfaktorn till en åttondel.

Om draken vaknar måste alla spelare i skattkammaren omedelbart lämna tillbaka samtliga skatter och flytta sin hjälte till valfri redan existerande intilliggande rumsbricka. Hjälten förlorar också 1–12 (slumpvis genererat) kroppspoäng.

Spelets vinnare

När alla spelare antingen har dött eller lämnat spelplanen jämförs de överlevandes skatter. Alla insamlade skatter har ett visst värde, och i slutet summeras de ihop för alla spelare. Den spelare vars skatter är värda mest vinner spelet.

3. Milstolpar

#	Beskrivning
1	Det finns datatyper för en spelplan med fyra startpunkter och en skattkammare i mitten. En testklass kan generera denna spelplan och visualisera den.
2	En generell klass för hjältar finns implementerad. Denna (super)klass ska innehålla de fält och metoder som är gemensamma för alla hjältar. Hjälten kan visualiseras på brädet och dess kroppspoäng är synligt bredvid brädet.
3	En klass som representerar rumsbrickor är implementerad. Klassen ska bland annat innehålla information om vilka sidor av kvadraten som är ”öppna”, det vill säga vilka håll som hjälten kan lämna rummet ifrån
4	En container innehållande de vanligaste rumsbrickorna är implementerad.
5	Turordningssytemet är implementerat, där en spelare får utföra en handling varje runda. Det ska också finnas en timer som räknar ner efter varje runda.

6	Spelaren kan välja en riktning att gå, dra en rumsbricka som placeras på vald position och flytta hjälten dit.
7	En klass som hanterar rumskort finns implementerad. Rumskorten ska dras och visas upp när hjälten flyttat sig till en ny ruta. Händelser relaterat till korten är ännu inte implementerat.
8	Event kopplade till rumskorten är till stor del implementerade. Funna skatter kan sparas och visas upp bredvid brädet. Funktioner för fällor är implementerade.
9	Stridsläget är implementerat.
10	En hjälte kan gå in i skattkammaren och ta två skattkammarkort.
11	Drakens funktionalitet är implementerad.
12	Djupare implementation av hjältar. Går att välja mellan fyra hjältar med olika namn och egenskaper innan spelet startar.
13	Spelare kan välja att lämna drakborgen när de befinner sig på en startpunkt.
14	En vinnare kan determineras när alla hjältar lämnat spelplanen
15	Rumsletning är implementerat.
16	Spelare får välja en magisk ring innan första rundan som kan hjälpa till i ett knivigt läge.
17	Dörrar och dörrkort är implementerade.
18	Ytterligare hjältar med speciella förmågor/vapen är möjliga att spela.
19	Magiska amuletter är möjliga att hitta via rumsletningskort och rumskort. Om spelaren väljer att hänga den runt halsen ska den aktiveras under specifika förhållanden (varierar med amulett).
20	Mer ovanliga brickor implementerade. Exempelvis vridrum, mörkerrum, bottenlös brunn.
21	En grundläggande datorspelare finns att spela mot. Datorspelaren följer en simpel, hårdkodad logik.
22	Mer speciella monster som vampyrfladdermöss, likätare och jättespindlar är implementerade. Strider med dessa följer specifika regler.
23	Resultat från spelomgångar kan loggas i en separat fil.
24	Alla hjältar från ursprungsspelet är implementerade och spelbara.
25	Underjorden och underjordskort är implementerade. Relevanta rumsbrickor och rumskort är tillagda.

26	Musik spelas upp under spelets gång
27	Ytterligare hjältar med fler specialförmågor är implementerade

4.Övriga implementationsförberedelser

Programmet ska bestå av en klass som hanterar spelbrädet och en klass som representerar hjältarna. Denna ska fungera som superklass för de individuella hjältarna som ärver fält och metoder som är gemensamt för alla hjältar. Vidare behövs en klass för rumsbrickor innehållande bland annat information om vilka sidor av brickan som är öppen. En annan klass ska hantera de olika kort som dras under spelets gång.

En klass ska hantera interaktionen med bräde, hjältar, och kort, medan en annan hanterar själva utritningen av spelet.

5.Utveckling och samarbete

Projektet utvecklas endast av en person. Arbetet kommer till största del ske utanför de schemalagda timmarna.

Slutinlämning

6. Implementationsbeskrivning

Detta kapitel beskriver de olika implementationer som gjorts och ämnar ge läsaren en uppfattning om hur projektet är strukturerat, vilka milstolpar som återstår att implementeras och vilka designbeslut som tagits under projektets gång.

6.1 Övergripande design

Denna sektion går igenom den funktionalitet som finns och hur den är uppdelad. I slutet av sektionen finns ett antal flödesdiagram som ger läsaren en uppfattning om hur programmet hanterar från spelaren.

6.1.1 Spelbrädet

Då Drakborgen är ett brädspel är en viktig del av spelet representation och manipulation av brädet. Denna funktionalitet hanteras av klassen Board. När ett Board-objekt skapas genereras en tvådimensionell matris av Brick-objekt. Board har ingen koll på hur ett Brick-objekt ser ut, utan håller endast koll på denna matris. Utseendet hos ett Brick-objekt finns istället lagrat i objektet själv, i form av en av SquareTypes. En SquareType är ett enkelt enum som säger vilken typ av ruta som finns på varje plats i en Brick. För att skapa en ny Brick använder sig Board av klassen BrickGenerator, vars enda syfte är att skapa ett Brick-objekt med önskat utseende. Board innehåller metoder för att placera en ny Brick i matrisen, samt för att lysa upp en bricka som markerats av användaren. Den senare anropar en metod i Brick klassen för att få tillbaka en Brick med ändrat utseende som läggs i matrisen med Brick-objekt. Board har också en metod för att hämta en SquareType hos en Brick på en given plats. Utöver upplysningsmetoden innehåller även Brick metoder för att rotera sitt utseende. Detta beror på att BrickGenerator skapar Brickor med ett visst utseende uppifrån, men beroende på åt vilket håll en hjälte rör sig på spelplanen kommer detta kräva att vissa Brickor roteras. Annars kan exempelvis en Brick med öppning till vänster ha en öppning till höger när den placerats på brädet.

Metoderna i Brick anropas exklusivt av Board, och metoderna i Board anropas av klasserna GameComponent och GameViewer.

6.1.2 Utritning av brädet

Utritningen av brädet hanteras av GameComponent, en rektangel i taget. Den använder sig av metoder från Board för att få storleken på spelbrädet samt den tidigare nämnda metoden som hämtar en SquareType på en viss plats av ett Brick-objekt. Genom att korrelera varje SquareType med en färg vet GameComponent vilken färg varje rektangel som ritas ut ska ha.

6.1.3 Slumpgenerering av kort och brickor

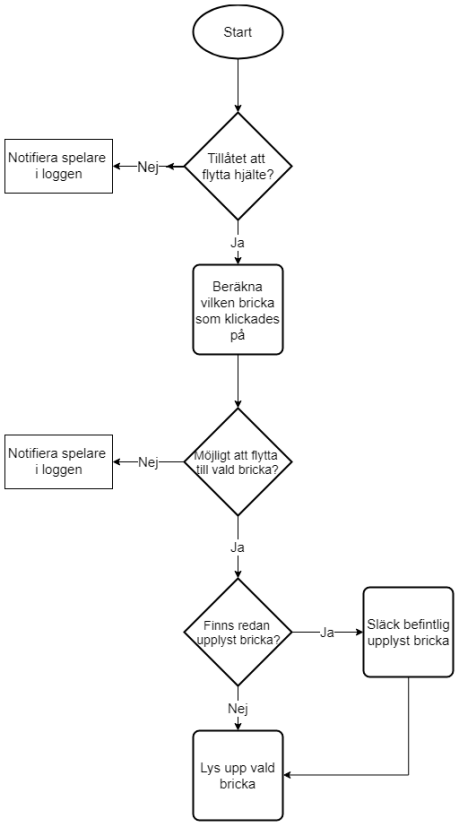
Under spelets gång kommer spelare att dra brickor och kort, därför är det viktigt att det finns ett sätt att slumpa fram dessa. Detta görs med hjälp av klasserna BrickGenerator och CardGenerator. Dessa klasser skapar vid konstruktion en lista av kort/brickor som blandas om, och har metoder för att dra dessa.

6.1.4 Användargränssnitt

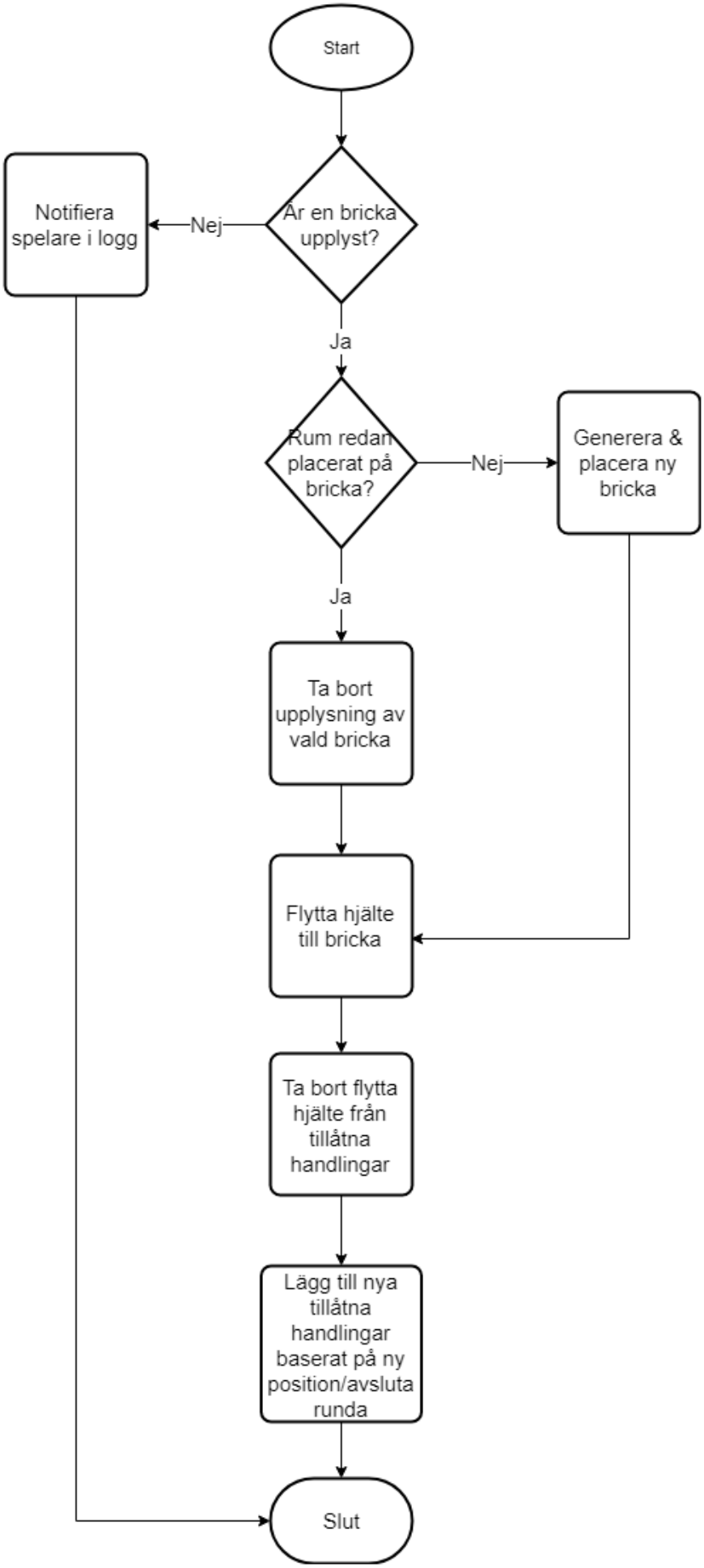
När spelet startar får varje spelare fylla i sitt namn och välja en hjälte att spela. Efter det går spelet in i sin primära state. Denna state är statisk och här väntar spelet på interaktioner från spelaren. Dessa interaktioner är:

- Musklick på spelbrädet
- Klick på rumsbrickssknappen
- Klick på rumskortsknappen
- Klick på rumsletningsknappen
- Klick på skattkammarkortsknappen

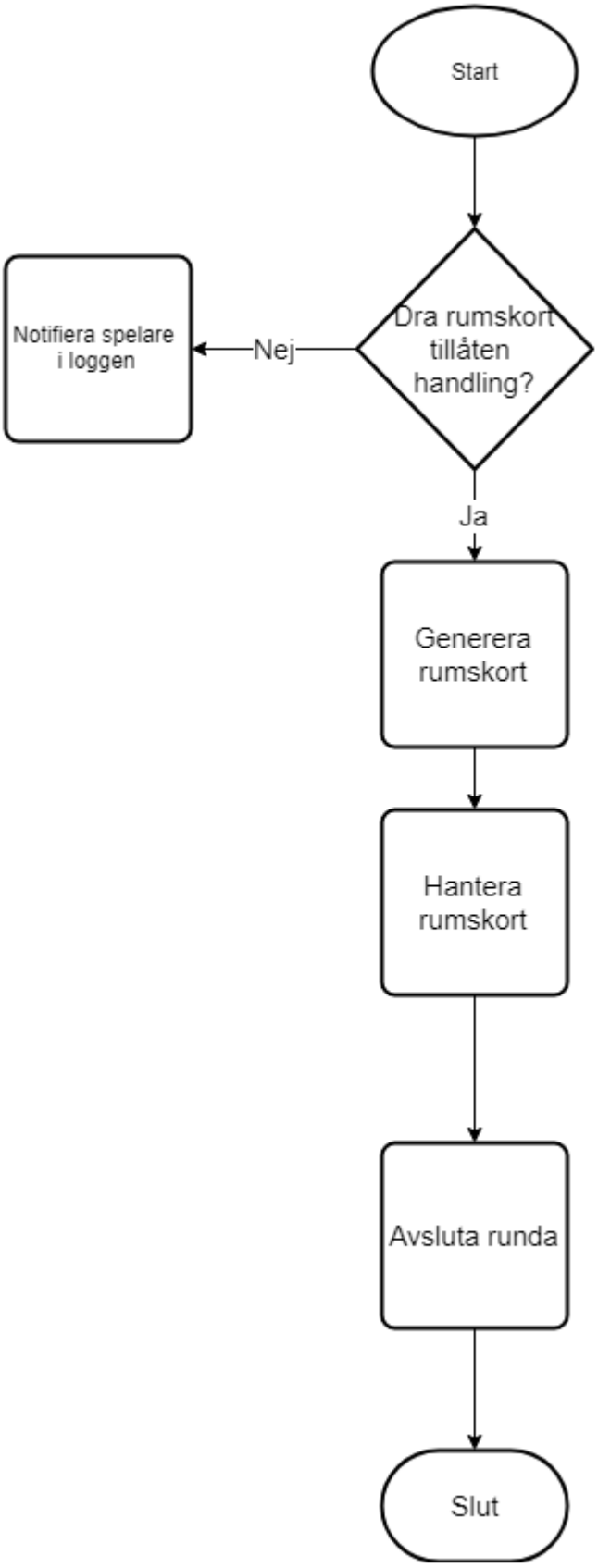
Denna funktionalitet ligger i klassen GameViewer och utgör större delen av kodbasen för detta projekt. GameViewer använder sig av en JFrame där alla komponenter fästs. Dessa komponenter är den tidigare nämnda GameComponent som ritar ut brädet, de fyra knapparna som nämns ovan samt en textlogg där spelaren får information om vad hen kan göra och vad som nyss har hänt. I samband med att en spelare drar ett rumskort eller rumsletningskort finns det en risk att stöta på ett monster. Då behöver spelaren slåss mot monstret och denna funktionalitet finns också i GameViewer. Även hantering av draken är implementerat i denna klass. För att bestämma vilka handlingar som är tillåtna finns det i GameViewer en lista med tillåtna handlingar. Denna lista uppdateras efter varje interaktion, samt när en spelares runda tar slut och en ny spelares runda börjar. Nedan redovisas fem stycken flödesdiagram som visar vad som händer vid de fem listade interaktionerna.



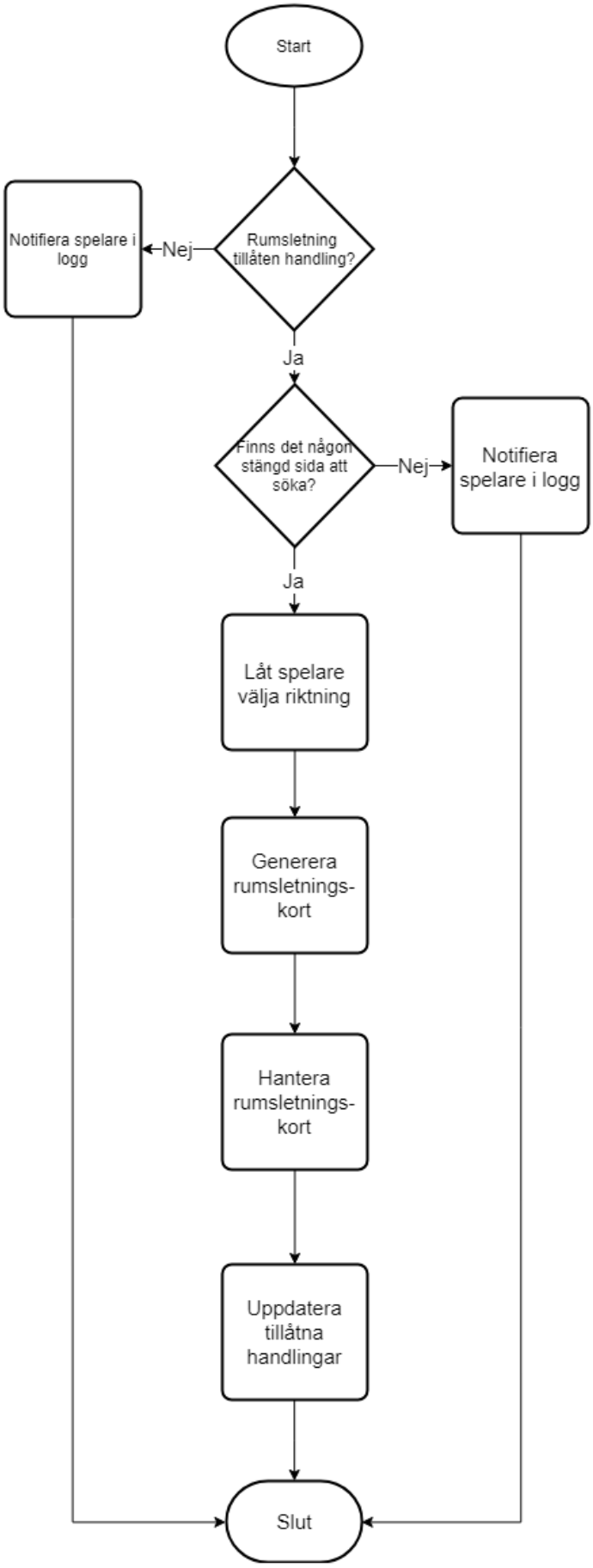
Figur 1 Vad som händer när spelaren klickar på brädet



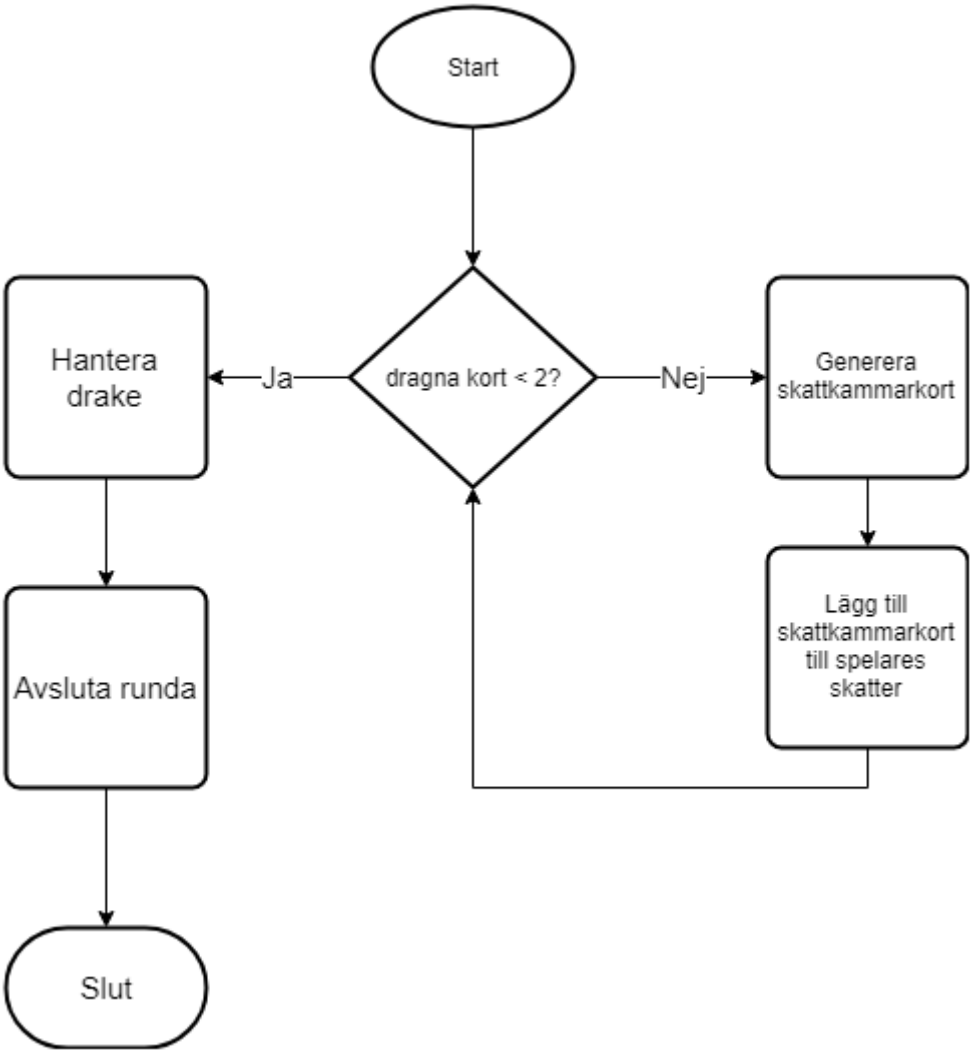
Figur 2 Hantering av klick på rumsbricksknapp



Figur 3 Hantering av klick på rumskortsknappen



Figur 4 Hantering av rumsletningsknappen



Figur 5 Hantering av klick på skattkammarknappen

Syftet med dessa flödesdiagram är att ge en överskådlig bild av hur användargränssnittet. Därför är exempelvis hantering av rumskort och rumsletningskort inte specificerad i diagrammen.

6.2. Milstolpar

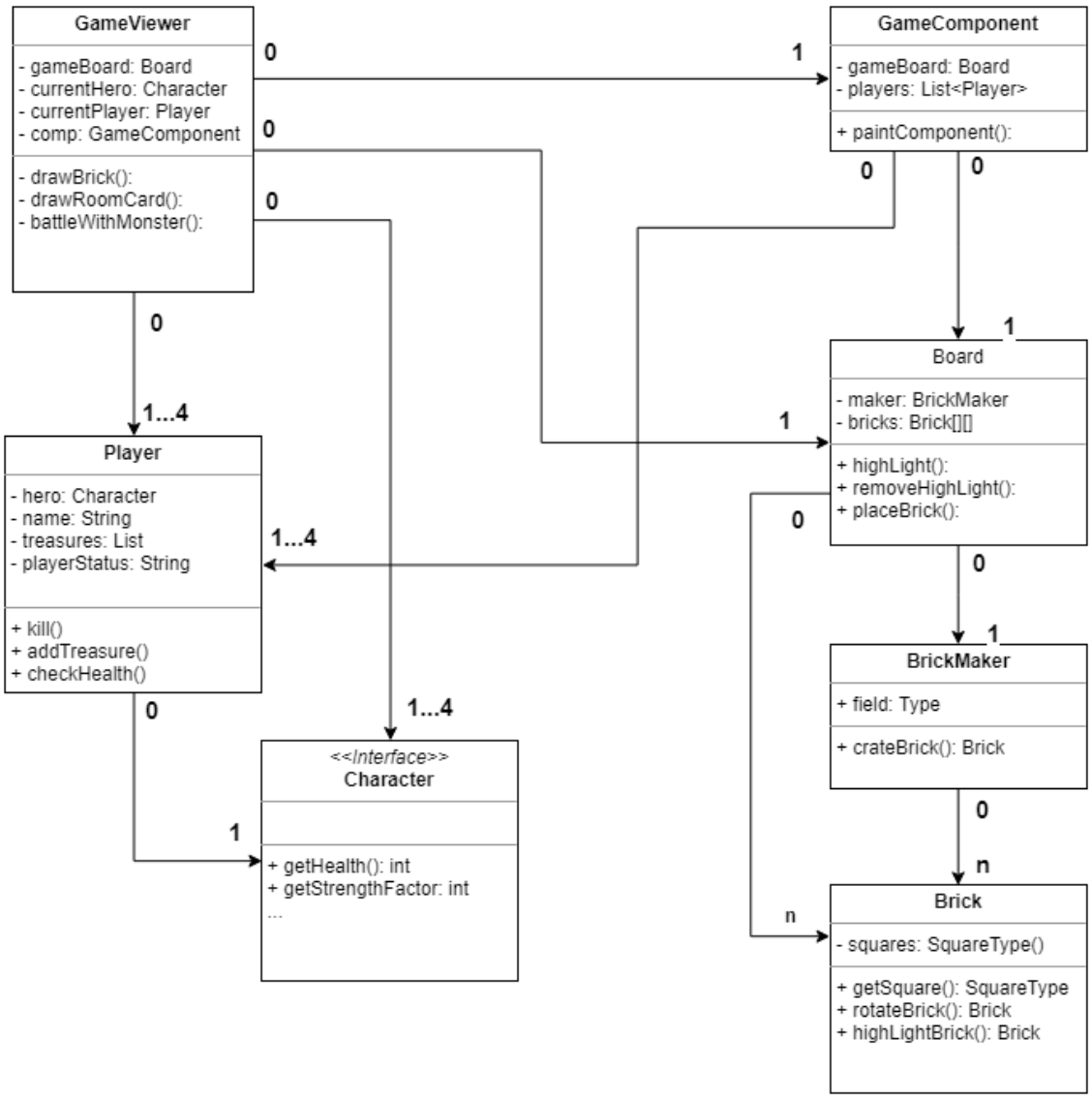
I detta kapitel redogörs vilka milstolpar som har implementerats och vilka som inte har hunnits med. Numreringen på milstolpar i listan nedan följer den från kapitel 3.

1. Spelplan är implementerad och kan illustreras. Milstolpe genomförd.
2. Generell klass för hjälten implementerad. Är synlig på brädet och dess egenskaper kan utläsas på sidan om spelplanen. Milstolpe genomförd.
3. Klass för rumsbrickor implementerad, innehåller dock inte information om ”öppna” sidor, utan denna koll görs användargränssnittet vid behov. Milstolpe delvis genomförd.
4. Container för rumsbrickor finns implementerad i klassen BrickGenerator. Milstolpe genomförd.
5. En spelare kan utföra ett visst antal handlingar, beroende på var hen befinner sig på spelplanen. En timer är synlig under spelplanen. Milstolpe genomförd.
6. Denna milstolpe är fullständigt implementerad.

7. Rumskort är implementerade och kan dras efter att en spelare har flyttat sig. Milstolpe genomförd.
8. Fällor och skatter är implementerade. Milstolpe genomförd.
9. Stridsläget är implementerat, om än på ett väldigt simpelt sätt. Milstolpe genomförd.
10. Milstolpe genomförd.
11. Milstolpe genomförd.
12. Milstolpe genomförd.
13. Milstolpe genomförd.
14. Milstolpe genomförd.
15. Milstolpe genomförd.
16. Milstolpe ej genomförd.
17. Milstolpe ej genomförd.
18. Milstolpe ej genomförd.
19. Milstolpe ej genomförd.
20. Milstolpe ej genomförd.
21. Milstolpe ej genomförd.
22. Milstolpe ej genomförd.
23. Milstolpe ej genomförd.
24. Milstolpe ej genomförd.
25. Milstolpe ej genomförd.

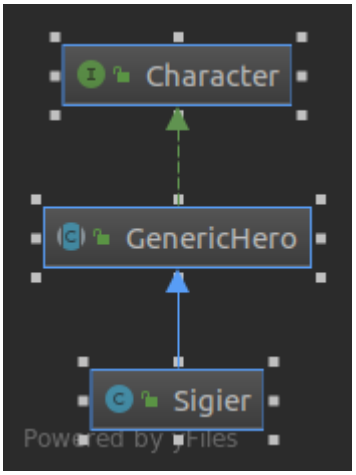
6.3. Dokumentation för programstruktur, med UML-diagram

I denna sektion beskrivs ett antal UML diagram. Syftet med detta är ge läsaren i utökad förståelse för projektets struktur.



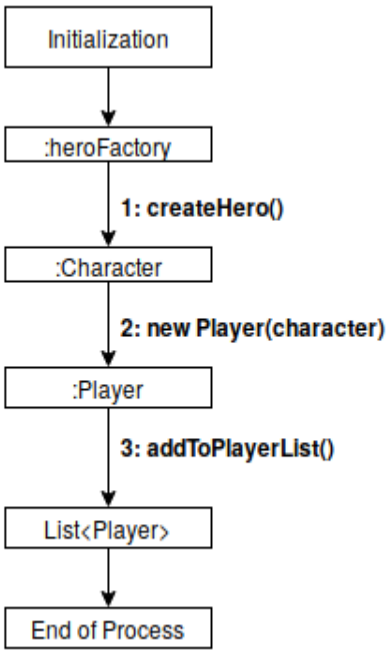
Figur 1 Översiktligt klassdiagram

Figur 1 visar ett klassdiagram över stora delar av systemet. Ur diagrammet kan vi se att klassen GameViewer äger kännedom om klasserna Board, GameComponent, Player samt interfacet Character. De nämnda klasserna känner däremot inte till GameViewer. Detta innebär att GameViewer kan interagera och utföra operationer på objekt av dessa klasser. Exempelvis använder GameViewer metoder från Board när något som hänt i användargränssnittet innebär att något ska ändras på spelbrädet.
Figur 2 visar ett klassdiagram över representationen av hjältar i spelet. I klassdiagrammet kan vi se att klassen Sigeir ärver från den abstrakta superklassen GenericHero, som i sin tur ärver från ett interface vid namn Character. Detta gäller för samtliga av de fyra hjälteklasserna i spelet, inte bara Sigier.



Figur 2 Klassdiagram för hjältar

I Figur 3 ser vi ett så kallat kollaborationsdiagram. Detta diagram ämnar visa flödet av information mellan objekt. I detta fall visar det processen av att skapa en ny hjälte, via interaktion med spelare, och koppla denna hjälte till spelaren.



Figur 3 Kollaborationsdiagram

6.4. Användning av fritt material

Utöver de klasser som ingår i Java 11 drar projektet användning av biblioteket apache.common.lang3. Från detta bibliotek har klassen ArrayUtils använts för att söka igenom listor för att se om de innehåller ett visst element. Projektet använder också klassen ImmutablePair för att representera en spelares skatter.

6.5. Motiverade designbeslut med alternativ

I denna sektion redovisas olika designbeslut som tagits under projektets gång. Var och ett av

dessa beslut jämförs med andra designbeslut som hade kunnat göras. Därefter reflekteras det kring vilket av dessa beslut som var det korrekta.

1. Det första designbeslutet var att bestämma hur spelplanen skulle representeras. Önskemålet var att ha ett bräde med 13x10 rutor, där varje ruta skulle kunna ha olika utseende. Samtidigt skulle det vara enkelt att hålla koll på vilken ruta en hjälte befinner sig och vilken typ av ruta det var. Lösningen blev att göra brädet till en matris med storlek 13x10 av Brick-objekt, där varje Brick har en intern struktur bestående av en 6x6 stor matris av SquareTypes. Detta gjorde att Board-klassen kunde hålla koll på vilken ruta som fanns på vilken plats, samtidigt som Board kan anropa metoder i Brick för att skicka information till GameComponent-klassen så att den kan rita ut varje SquareType. Ett alternativ hade varit att låta Board ha en 13*6x10*6 stor matris av SquareTypes. Board hade enkelt kunna skicka information till GameComponent för utritning, men det hade resulterat i att någon klass hade behövt räkna ut vilken ruta som varje hjälte befinner sig på utifrån matrisen av SquareTypes. Det hade också behövts någon annan container för att hålla koll på vilken typ av rumsbricka som fanns på varje ruta. Den valda lösningen bedömdes som mer effektiv då det blev avsevärt mycket enklare att hålla koll på vilken ruta man befann sig i.

2. För att kunna flytta en hjälte åt något håll måste den befintliga rumsbrickan ha en öppning åt detta håll. En koll på vilka sidor som var öppna gjordes i GameViewer när en bricka skulle lysas upp. Ett annat alternativ hade varit att lägga ett fält i Brick-klassen som innehöll information om vilka sidor som var öppna. Detta var mer i enlighet med vad som skrivits i milstolpe 3 och hade definitivt varit en bättre implementation, då det hade flyttat funktionalitet som rör en Brick till Brick-klassen, istället för att ha den i användargränssnittet.

3. För att en spelare ska kunna slåss med ett monster var kravet att skulle finnas tre attacker, där varje attack besegrar exakt en annan. Det vill säga att stridssystemet skulle implementera en form av sten-sax-påse funktionalitet. För att lösa detta skapades ett Enum med de tre attackerna med en abstrakt metod beats som tar en annan attack som indata. Denna metod, som skiljer sig för varje enum, returnerar en boolean som säger ifall attacken besegrar attacken som skickades med (i enlighet med reglerna som beskrivits i kapitel 2). På så sätt kunde GameViewer slumpa fram en monster attack och kalla på beats för spelarens attack och monstrets attack för att se vilken som vann. En annan lösning hade varit att istället lägga logiken direkt i GameViewer. Detta hade dock lagt funktionalitet i användargränssnittet som inte var direkt kopplat till användarinteraktion, och bedömdes därför som en sämre design.

4. I Drakborgen finns ett antal olika hjältar, där varje hjälte har ett antal konstanta fält som bestämmer dess egenskaper och hur den ska ritas ut på brädet. Jag har valt att skriva en klass kallad heroFactory, vars metod createHero, som endast tar en sträng som indata, skapar en av de tillgängliga klasserna utifrån denna sträng. I denna klass deklareras alla hjältars konstanta värden och skickas sedan med till respektive hjältes konstruktor. Ett annat alternativ hade varit att istället deklarera dessa fält i respektive hjälte-klass och låta dess konstruktor ta noll argument. Då hade nya hjältar kunnat skapas direkt i main funktionen istället för via factory-klassen. Anledningen till mitt val var dels att det höll respektive hjälte-klass liten och lättförståelig. Dessutom möjliggjorde det nästa designbeslut, som gäller hierarkin bland hjälteklasserna.

5. För att undvika redundanta metoder i hjälte-klasserna skrev jag en abstrakt klass, GenericHero, som alla hjältar ärver ifrån. Denna abstrakta klass innehåller samtliga

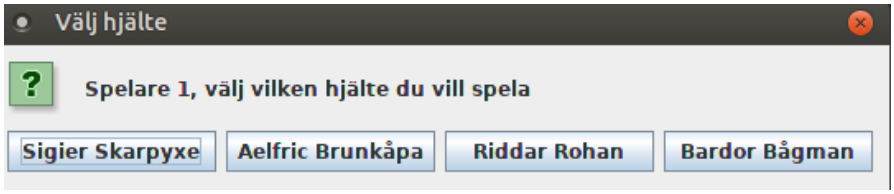
metoder som är gemensamma för alla hjältar, som till exempel getters för hjältens egenskaper. Utöver det använde jag mig av ett interface, Character, vars funktioner implementeras i GenericHero. Ett annat alternativ hade varit att låta varje hjälteklass implementera funktionerna från mitt interface, och skippa den abstrakta klassen. Detta hade dock resulterat i att varje hjälteklass hade haft identiska metoder, vilket inte är rekommenderbart. Därför är det bättre med en abstrakt klass.

6. För att kunna simulera kortdragning måste det finnas någon funktionalitet som slumpmässigt genererar ett card-objekt. Detta gjordes genom att skapa en lista med ett specificerat antal av varje kort. Detta gjordes för alla olika typer av kort som kan dras under spelets gång. Ett annat alternativ hade varit att bara generera ett slumpmässigt tal och låta olika intervall avgöra vilket kort som ska genereras. Ett problem med detta är dock att sannolikheten för att dra ett visst kort måste minska efter att ett sådant kort dragits. Detta skulle innebära att intervallen behövde uppdateras efter varje draget kort, vilket bedömdes som onödigt krångligt, och inte värt den eventuella minneskapacitet som sparades.
7. När en ruta på spelbrädet ska lysas skapas ett ny Brick-objekt med Squares av typen HIGHLIGHTED, och om en spelare vill markera en redan placerad bricka måste det på något sätt sparas undan hur många gånger brickan har roterats, annars kommer brickan, när den inte längre är upplyst, placeras tillbaka i sin ursprungliga form. Detta kan ge helt fel utseende på spelbrädet. För att hantera detta skapade ett privat fält i Brick som sparade hur många gånger en bricka roterats. Detta fält uppdaterades vid anrop till rotate metoderna i Brick och användes vid återskapande av en bricka som tidigare varit upplyst. Ett annat sätt att lösa detta problem hade varit att temporärt spara undan den befintliga brickan innan upplysning. När brickan sedan skulle återskapas hade det räckt att lägga tillbaka den undansparade brickan, som redan roterats första gången den placerats på spelbrädet, i matrisen i Board. Denna implementation hade varit bättre att göra, då det betyder att fältet med antal rotationer inte hade behövts, och det hade också behövts färre anrop till rotate metoderna i Brick.
8. För att bestämma hur en bricka ska se ut måste det någonstans finnas funktionalitet för att skapa en Brick med ett visst internt utseende. Detta valdes att göra med hjälp av klassen BrickMaker som håller koll på de olika utseenden som en Brick kan ha. Den publika metoden createBrick(), som tar en BrickType som argument, används för att skapa ett nytt Brick-objekt. Ett alternativ till detta hade varit att lägga denna funktionalitet direkt i Brick-klassen. Då hade konstruktorn i Brick bara behövt ta emot en BrickType och sedan bygga upp den interna matrisen av SquareTypes baserat på denna BrickType. Nackdelen med detta är dock att det skulle resultera i att varje Brick skulle äga kännedom om utseendet hos alla typer av Bricks, och inte bara sin egen. Detta följer inte riktigt objektorienterad praxis och bedömdes som en sämre lösning.

7. Användarmanual

Spelet startas med hjälp av klassen Game, och dess main metod.

Vid start av spelet visas en prompt där en användare får fylla i hur många spelare som ska spela. Därefter får varje spelare fylla i sitt namn och välja en av de fyra hjältar som finns implementerade. Prompten för att välja hjälte visas i Figur 1.



Figur 4 Prompt för att välja hjälte

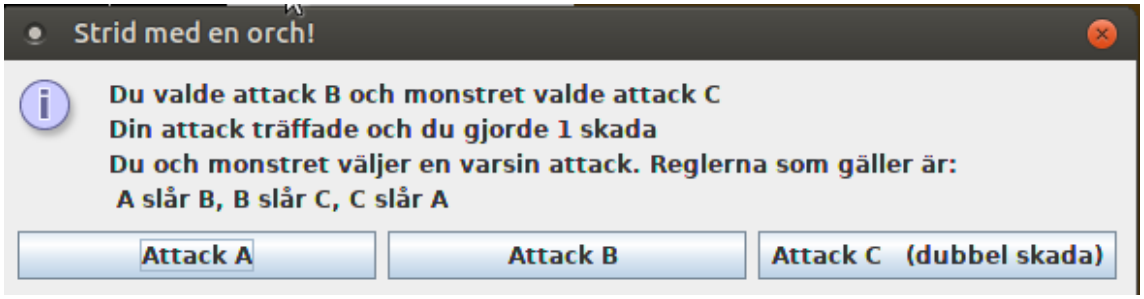
Figur 5 Spelets huvudfönster med spelbrädet till vänster och användargränssnitt till höger

När alla spelare har valt hjälte visas användargränssnittet som låter användare spela spelet, med spelbrädet till vänster och knappar/informationsrutor till höger. Detta går att se i Figur 2. Spelet går till genom att spelare väljer vilka handlingar de vill utföra genom att klicka på knapparna eller genom att klicka på brädet. Alla hjältar startar på någon av de fyra startrutorna i hörnen (markerade gröna).

I början av en spelares runda är det alltid tillåtet att flytta hjälten. Detta görs genom att klicka på någon av de rutor som angränsar till den ruta som spelarens hjälte befinner sig på och som det finns en öppning till (illustreras av ljusgrå färg, till skillnad från en vägg som illustreras av en mörkgrå färg). När en ruta har blivit blåmarkerad är den markerad, och för att flytta till den rutan klickar spelaren på knappen med texten ”dra rumsbricka” (eller ”flytta hjälte” om det redan finns en bricka på den rutan). Spelarens hjälte har då förflyttat sig till den nya brickan.

Om det inte finns någon öppning åt det håll en vill förflytta hjälten går det att söka efter en lönndörr i rummet. Spelaren klickar då på knappen med texten ”dra rumsletningskort”. Spelaren får då välja vilken riktning hen vill söka i, detta avgör var spelaren kommer att hamna om rumsletningen är framgångsrik och spelaren finner en lönndörr i rummet. Om spelaren hittar en lönndörr räknas det som att flytta hjälten som vanligt, och rundan fortsätter. Om spelaren däremot inte finner någon lönndörr är hens runda över efter att rumsletningskortets innehåll har behandlats.

När en hjälte har förflyttat sig till en ny ruta, och den rutan inte är en startruta eller skattkammarruta, måste spelaren dra ett rumskort. Detta görs genom att klicka på knappen med texten ”dra rumskort”. En inforuta kommer då att dyka upp och berätta vad som finns i rummet. Spelaren får följa de instruktioner som dyker upp för att hantera rummets innehåll. Detta kan vara fällor, skatter eller monster. Om rummet innehåller ett monster, antingen genom ett överfall eller genom att stöta på monstret som vanligt, initieras ett stridsläge. Detta visas i Figur 3. Det är då upp till spelaren att välja vilken attack hen vill använda. Efter varje val jämförs spelarens val av attack med monstrets val av attack. Resultatet är antingen att monstret tar skada, eller att hjälten tar skada eller att båda tar skada. Striden är över när antingen hjältens eller monstrets kroppspoäng når 0. Spelaren vet inte hur många kroppspoäng ett monster har förrän det har besegrats. När rumskortets innehåll har hanterats är spelarens tur över.



Figur 6 Prompt för stridsläget, resultatet av tidigare rond visas längst upp

Om hjälten befinner sig i skattkammaren, antingen i starten av rundan eller efter att ha förflyttat sig, kan spelaren klicka på knappen med texten ”dra skattkammarkort”. Spelaren kommer då att dra två skattkammarkort som innehåller någon form av skatt. Dessa skatter läggs till i spelarens pott. Men när skattkammarkorten har dragits måste det kollas om draken vaknar eller inte. Första gången kollen görs är chansen att draken vaknar 1/8. Nästa gång är chansen 1/7 och så vidare tills det inte längre finns någon hjälte i skattkammaren. Då återställs chansen till 1/8 igen. Om draken vaknar tar alla hjältar som befinner sig i skattkammaren mellan 1 och 12 skada och förflyttas ut från skattkammaren. Spelarna förlorar också alla sina skatter från skattkammaren.

Spelet går ut på att lämna Drakborgen med så mycket guld som möjligt. En kan endast lämna drakborgen när en befinner sig på en startruta. Detta görs genom att klicka på knappen med texten ”Lämna drakborgen”. Denna knapp är endast synlig om spelarens hjälte befinner sig på en startruta. Om en hjältes kroppspoäng når noll, eller om hjälten fortfarande befinner sig på spelplanen när soltimern som visas under spelbrädet når noll, har spelaren förlorat och är inte längre med i spelet. Vinnaren koras när alla spelare antingen har lämnat eller dött och vinner gör den som tagit sig ut med skatter med högst värde. I inforutan direkt till höger om spelbrädet (se Figur 2) kan en spelare se hur stort värde dennes befintliga skatter är värda. Där går det också att se hur många kroppspoäng hjälten har kvar. Inforutan visar alltid information relevant till den hjälte vars tur det är att spela.

8. Utvärdering och erfarenheter

- *Vad gick bra? Mindre bra?*
  - Det mesta under projektets gång har gått bra. Det största misstaget som gjordes var att inte kolla upp deadlines i god tid. Jag upptäckte inte deadlinen för rapporten förrän 5 dagar innan, och då hade jag inte skrivit något på den. Borde definitivt ha skrivit på rapporten allt eftersom, istället för i slutet.

- *Vilket material och vilken hjälp har ni använt er av? Har ni gått på föreläsningar? Läst boken? Letat på nätet? Gått på handledda labbar? Ställt många frågor? Vad har ”hjälpt” bäst? Vi vill gärna veta för att kunna vidareutveckla kurs och kursmaterial åt rätt håll!*
  - Jag har nästan endast använt mig av nätet för att finna information. Har gått på några labbpass för att ställa frågor.
  - *Har ni lagt ned för mycket/lite tid?*
    - Har lagt ungefär så mycket tid som förväntat, beroende på hur man räknar. Om projektet är 80 timmar/person och jag som arbetat ensam förväntas lägga det dubbla har jag nog lagt mindre tid än förväntat.
  - *Vad har varit mest problematiskt, om man utesluter den programmeringstekniska delen? Alltså saker runt omkring, som att hitta ledig tid eller plats att vara på.*
    - Mest problematiskt för mig har varit att kombinera detta projektet med kandidatprojektet som jag läst parallellt, då en del deadlines har legat på samma dag. Detta beror dock på att jag läst denna kurs tidigare år men inte

gjort projektet.

- *Vilka tips skulle ni vilja ge till studenter i nästa års kurs?*
    - Tips skulle vara att skriva på slutrapporten allt eftersom man implementerar programmet, samt att noga tänka över val av projekt så att det man väljer att göra är rimligt att utföra.