NTI Gymnasiet
Sundsvall

Gymnasiearbete
HT 2019

Isak Einberg

Handledare:
Emma Selander, Jennie Udd

Röststyrt Geografispel

Ett lärorikt spel i Java som styrs med rösten

Abstract

This paper contains information regarding the development of a voice-controlled computer game made with the Java programming language that, unlike most games, prioritizes educating the player rather than being as enjoyable as possible.

The purpose with this report is to explain the process of developing a voice-controlled game and learning how speech recognition can be used to develop voice control. The goal was to create an educative flag and geography quiz that makes use of advanced speech-recognition in order to let a player guess different properties of european flags and nations.

The project covered in this report was successful and the result is a game that is playable using speech as the sole input. The game includes every planned feature as well as every european nation. It is to be showcased in a quiet room where visitors will be able to try the game out for themselves.

Innehåll

1. Inledning

1.1. Bakgrund

Jag tycker själv att taligenkänning och röststyrning är intressanta områden. Från dessa intressen utvecklades de första tankarna att implementera taligenkänning och röststyrning till någon form av hårdvara eller mjukvaruapplikation. Eftersom jag även har ett intresse för geografi och flaggor togs beslutet att skapa ett spel där spelaren prövas inom dessa områden, detta genom att fullständigt styra spelet med rösten.

1.2. Syfte och mål

Målet med arbetet är att programmera ett röststyrt spel med programspråket Java. Spelet ska vara lärorikt och ska förbättra spelarens kunskaper om geografi och flaggor, detta genom att presentera spelaren med silhuetter av olika länder. Olika mycket fakta om landet visas beroende på vald svårighetsgrad, och spelaren ska med sin röst berätta för programmet vilken information som saknas om landet. Spelaren behöver bidra med all information som saknas innan den får gå vidare till nästa land. Exempelvis behöver kanske spelaren ange ett lands namn och huvudstad innan den får gå vidare till nästa land.

Spelet kommer till viss del efterlikna geografispelet Seterra (www.seterra.com). Spelet kommer att skilja sig utseendemässigt från Seterra, men spelets mål kommer att vara mycket likt då målet i Seterra är att gissa rätt länder eller flaggor. Skillnaden kommer vara att spelaren i mitt spel kommer få gissa på flera saker samtidigt (t.ex. både flagga, landsnamn och huvudstad) istället för en sak åt gången, samt att spelet kommer att kontrolleras med röststyrning.

1.3 Frågeställning

- Hur kan taligenkänning implementeras i ett Javaprogram?
- Hur kan taligenkänningen användas för att skapa röststyrning?

1.4 Avgränsning

Spelets utseende och användarvänlighet kommer inte prioriteras, utan fokus kommer ligga på att skapa alla spelets funktioner samt att få dessa att fungera så problemfritt som möjligt med spelets taligenkänning. Spelet kan därför sakna estetiskt innehåll i form av t.ex. animationer och en vacker design.

1.5 Koppling till examensmål

Utveckling av kunskaper om och färdigheter i teknik och teknisk utveckling.[1]
Under arbetets gång kommer kunskaper om både programmering och tekniska hjälpmedel utvecklas. Även kunskaper om hur taligenkänning fungerar och kan implementeras i applikationer kommer att utvecklas.

Utveckling av kunskaper (..) med fokus på tekniska processer.[2]
Kunskaper inom taligenkänningsprocesser kommer utvecklas under arbetets gång. Även kunskaper gällande de tekniska processer som rör objektorienterad programmering kommer att utvecklas.

Utveckling av förmåga att använda digital teknik.[3]
Förmågan att använda digital teknik utvecklas då spelet programmeras med programspråket Java i en integrerad utvecklingsmiljö på en dator.

Ge kunskaper om projektarbete och färdigheter i att arbeta i projekt.[4]
Detta arbete är ett projekt, och under dess gång utvecklas kunskaper och färdigheter i planering, arbete och utvärdering.

Utveckla förmåga att analysera och förstå tekniska system.[5]
Under arbetets gång utvecklas dessa förmågor bland annat genom skapande, analys och revidering av kodade funktioner, samt implementation av funktioner från bibliotek i programvara.

---

[1] https://www.skolverket.se/undervisning/gymnasieskolan/laroplan-program-och-amnen-i-gymnasieskolan/gymnasieprogrammen/program?url=1530314731%2Fsyllabuscw%2Fjsp%2Fprogram.htm%3FprogramCode%3DTE001%26tos%3Dgy&sv.url=12.5dfee44715d35a5cdfa9295#anchor_1 (Hämtad 2019-09-04)
[2] Ibid.
[3] Ibid.
[4] Ibid.
[5] Ibid.

2. Genomförande

2.1. Planering

Arbetet ska inledas med att pröva och välja ut en extern tjänst eller Javabibliotek för taligenkänning. Olika tjänster kommer att prövas och jämföras, och slutligen kommer en av dessa väljas ut för att användas i spelet. Funktioner från den valda tjänsten/biblioteket kommer sedan undersökas för att ge en bättre förståelse över hur taligenkänningen ska bli en del av spelet. Ett grundläggande Javaprogram kommer skapas för att inledningsvis testa att taligenkänningen fungerar som tänkt.

Efter det är bekräftat att taligenkänningen fungerar som tänkt inleds kodandet av spelets funktioner och gränssnitt. Taligenkänningen implementeras från början och kommer finnas med under hela spelets utveckling eftersom det är spelets kärna.

Med spelets grundläggande funktioner och gränssnitt färdiga kommer fokus ligga på att implementera fler funktioner som exempelvis sparade resultat och olika svårighetsgrader med olika innehåll. Spelets innehåll kommer därefter utökas efter hur mycket tid som finns tillgänglig.

Innan projektets slut testas spelet. Den resterande tiden av projektet spenderas till att åtgärda eventuella problem som dyker upp under testningen.

Fokus kommer alltså i första hand att ligga på att utveckla själva spelet och dess funktioner. I andra hand kommer fokus att ligga på att utöka spelets innehåll i form av fler länder med flaggor. Från början kommer endast ett fåtal länder att inkluderas i spelet, och fler läggs till om det finns tid över då spelets funktioner är färdiggjorda.

2.2. Tidsperspektiv

Tiden dedikerad till varje moment fördelas enligt ett Gantt-schema. Schemat inkluderar inte endast spelutvecklingens delmoment, utan innefattar även skrivning av projektplan, rapport och loggbok. Om ett delmoment kräver mer tid än planerat revideras Gantt-schemat, då för att bättre anpassa projektets upplägg.

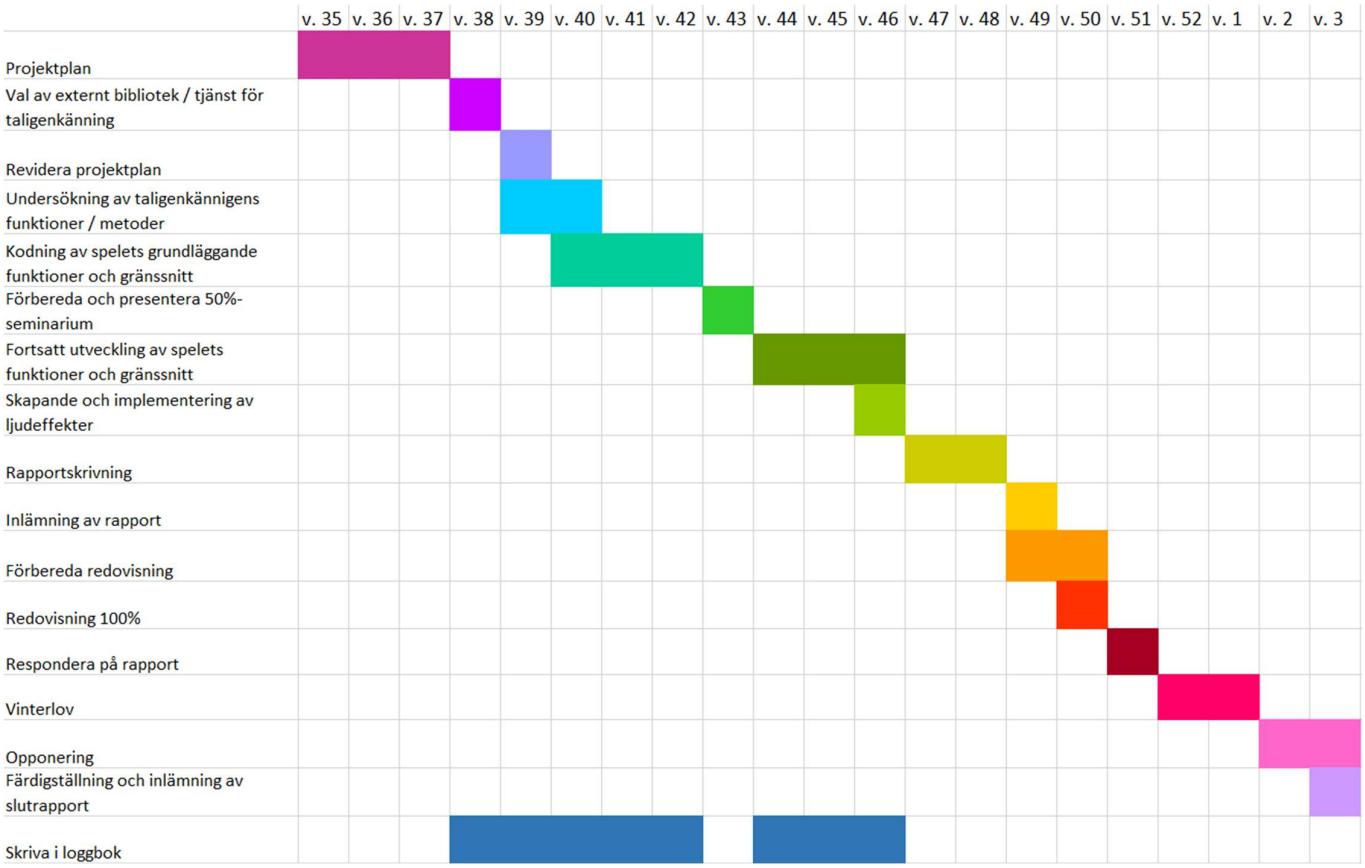


Bild 1. Gantt-schema.

2.3 Metodbeskrivning

Arbetet kommer utföras på en bärbar dator med operativsystemet Windows 10 samt en stationär dator med samma operativsystem och verktyg. Själva spelet kommer kodas i programspråket Java i den integrerade utvecklingsmiljön IntelliJ av JetBrains. Ett programspråk är ett språk som låter människor använda kodord för att ge instruktioner till en dator. Koden skrivs i den integrerade utvecklingsmiljön där den sedan kompileras, det vill säga översätts till maskinkod som datorn handlar efter.

Taligenkänningsfunktioner implementeras från en extern tjänst eller bibliotek i form av färdiggjorda *klasser*, vilket kort sagt är olika samlingar av kod. Även klasser inbyggda i själva Javaplattformen kommer användas till spelet.

Uppgifter och bilder relaterade till Europas flaggor och länder hämtas från webbplatserna Wikipedia.com och Silhouettegarden.com. Dessa bilder redigeras med bildmanipuleringsverktyget Paint.NET.

Som resurs för kodningen av spelet används Stackoverflow.com som hjälpmedel för att hitta lösningar till eventuella problem med koden.

2.4 Arbetet

2.4.1 Förberedelser

Arbetet inleddes med att pröva och jämföra två olika taligenkänningsbibliotek/tjänster, Sphinx-4 och Google Cloud Speech-to-Text (GCS). Det som jämfördes var kostnad, begränsningar och språkstöd. Sphinx-4 är gratis och arbetar lokalt på datorn (d.v.s. kräver ingen internetanslutning), men stödjer inte det svenska språket. Däremot låter Sphinx-4 en att "skapa" egna ord genom att placera stavelser i olika ordningar. GCS stödjer det svenska språket men kräver en internetuppkoppling och har en rörlig kostnad som ökar ju mer tjänsten används. GCS uppfattar engelskt tal extremt väl, men då det gäller svenskt tal presterar tjänsten inte lika väl.

GCS valdes över Sphinx-4 med anledningen att det stödjer det svenska språket och uppfattar tal mycket bättre än Sphinx-4. När ett Google-konto registreras för GCS tilldelas 300 amerikanska dollar som får användas till tjänsten, vilket är mer än tillräckligt för hur mycket tjänsten kommer att användas i detta arbete. Kostnaden är 0,006 amerikanska dollar per 15 sekunder som tjänsten används.

Efter att taligenkänningstjänsten valts ut implementerades den i ett nyskapat Java-projekt i IntelliJ. Klasser från taligenkänningens bibliotek infogades automatiskt till projektet genom att i IntelliJ:s pluginbibliotek söka efter pluginet Google Cloud, installera det och att sedan logga in i det med det Google-konto som tjänsten aktiverats för. Därefter hämtades en nyckel-fil från konsolsidan på Google Clouds webbplats (https://console.cloud.google.com/), vilken sedan sparades i projektets filmapp. Filsökvägen för nyckeln lades sedan in i IntelliJ, vilket låter GCS hitta filen samt vilket ger Java-projektet åtkomst till tjänsten.

Med GCS implementerat och aktiverat kodades en grundläggande testapplikation med Javas inbyggda grafiska gränssnitt Swing. Testapplikationen inkluderade olika former som färgades på kommando. Exempelvis färgades en triangel röd om taligenkänningen uppfattade meningen "färga triangeln röd". Programmet fungerade genom att GCS uppfattar tal och ger programmet en textsträng, d.v.s. en *String*, som sedan kollas med den inbyggda metoden *String.contains(X)*, där *String* är den textsträng som kollas samt där *X* är det innehåll som söks.

Då testapplikationen bevisade att GCS kunde användas för röststyrning inleddes kodningen av själva geografispelet.

2.4.2 Spelets grundläggande gränssnitt och funktioner

Inledningsvis skapades klassen *Land* med följande fält:

- *namn*, *huvudstad* och *flaggtyp* av typen String (textsträng)
- *flaggtyp* och *världsdel* av typen int (heltal)
- *flaggKont*, *silhuett* och *flaggFull* av typen ImageIcon (bild)

Klassen fungerar som en struktur för länderna i spelet, och dess fält är attributer relaterade till landet. När ett objekt av typen Land initieras tilldelas alla fält sina angivna värden. Ett lands bilder lästes vid det här laget in från den enskilda mappen *bilder* i Javaprojektet, något som kommer att ändras senare. För att enkelt kunna ange korrekt världsdel och flaggtyp skapades konstanter med namn som motsvarar de olika typerna/världsdelarna. Se Bild 2.

```
//Nyckelordet "final" hindrar variabelns värde från
//att ändras efter att den har initialiserats.
//Variabeln blir alltså en konstant
final static int
        FLAGGTYP_KORS = 0,
        FLAGGTYP_TRIKOLOR = 1,
        FLAGGTYP_BIKOLOR = 2,
        FLAGGTYP_ANNAN = 3,
```

Bild 2. Konstanter för olika flaggtyper.

För att kunna testa spelets funktioner under utvecklingen skapades länderna Sverige och Norge. Några fler länder behövdes inte till en början, då utvecklingen av själva spelet först prioriterades över spelets innehåll.

Den grundläggande röststyrningen till spelet skapades genom att låta programmet undersöka innehållet i den textsträng som taligenkänningen uppfattar, detta på samma vis som för testapplikationen till spelet. Varje gång taligenkänningen uppfattar ett ord kallas metoden *check()* i klassen Logik, som sköter hela spelets röststyrning. Flertalet variabler av typen boolean skapades, vilka representerar den information som visas i spelet. *huvudstadrev = true* tyder exempelvis på att huvudstaden är korrekt gissad samt att den ska visas för spelaren. Delen "rev" i namnet kommer från engelskans *reveal*. Alla dessa booleans har inledningsvis värdet *false*, då informationen vid spelets början inte är avslöjad.

För att gissa rätt huvudstad för ett land jämförs den huvudstad som spelaren säger med resterande huvudstäder i spelet. Gissar spelaren rätt får huvudstadrev värdet *true*. Nämner spelaren en annan huvudstad än den korrekta tappar den ett "liv". Livsystemet var vid det här laget av spelets utveckling endast en int, *liv*, vilken värdet "1" subtraherades från vid varje felaktigt svar. När *liv* fick värdet 0 förlorades spelet. För att tappa ett liv kallades metoden *felsvar()*. Bild 3 visar den kodsnutt ur röststyrningsmetoden (*check()*) som behandlar huvudstäder.

```
//Kollar om spelaren redan har gissat rätt på huvudstaden.
//rev = revealed = avslöjad, d.v.s att spelaren har listat ut vad det är.
if (!Spel.huvudstadrev) {

    //"in" är en String (text) som innehåller de ord som taligenkänningen
    // tror att spelaren sade.
    //"in.contains(x)" kollar om Stringen "in" innehåller texten "x".
    // "huvudstaden" innehåller t.ex. "staden".
    if (in.contains("stad")) {

        boolean saEnHuvudstad = false;
        String denSomSades = "";

        //Kollar om spelaren sa namnet på någon huvudstad genom att jämföra
        //med ALLA länders huvudstäder.
        for (int i = 0; i < Konstanter.stadArray.size(); i++) {
            if (in.contains(Konstanter.stadArray.get(i))) {
                saEnHuvudstad = true;
                denSomSades = Konstanter.stadArray.get(i);
                break;
            }
        }

        //Om användaren sa rätt huvudstad
        if (in.contains(Spel.land.huvudstad)) {
            Spel.huvudstadrev = true;
            korrektSvar( typ: "huvudstadrev");
        }

        //Om användaren sa en huvudstad, fast inte rätt huvudstad
        else if (saEnHuvudstad) {
            boolean sagd = false;
            for (Land l : tidigare) {
                if (l.huvudstad.equals(denSomSades)) {
                    sagd = true;
                    break;
                }
            }
            if (!sagd) {
                Spel.huvudstadrev = true;
                felsvar( s: "Fel huvudstad. Du sa '" + denSomSades +
                        "' Rätt svar: " + Spel.land.huvudstad);
            }
```

Bild 3. Uppfattad huvudstad kontrolleras.

Den grafiska delen av spelet skapades genom att skapa en klass, *Frame*, som fungerar som ett utskott av Swing-klassen JFrame. Denna JFrame fungerar som ett fönster för hela spelets grafiska innehåll, och inkluderar bland annat alla knappar som visas på huvudmenyn. Bilder, texter och annat grafiskt innehåll ritas upp i detta fönster med klassen *Rita*, som fungerar som ett utskott av Swing-klassen JComponent, och implementerar klassens metod *paintComponent()*. *paintComponent()* bestämmer det grafiska innehåll som ska ritas upp i

fönstret, detta genom att jämföra olika villkor i metoden. Det grafiska innehåll som anges i metoden ritas upp då *Rita.repaint()* kallas. Denna metod måste kallas varje gång spelets grafiska innehåll behöver uppdateras, då det är i metoden *paintComponent()* som allt grafiskt innehåll exklusive huvudmenyns innehåll anges. Huvudmenyns innehåll finns i Frame-klassen, och visas upp då klassen initieras. Bild 4 visar ett avsnitt ur klassen *Rita*, där värdet av boolean-variabeln *saAvsluta* avgör om bilden *villAvslutaBild* eller bilden *avslutaBild* ska ritas upp.

```
public static class Rita extends JComponent {
    @Override
    protected void paintComponent(Graphics g2) {
        Graphics2D g = (Graphics2D) g2;
        if (saAvsluta) {
            g.drawImage(villAvslutaBild,  x: 1640,  y: 0,  observer: this);
        } else {
            g.drawImage(avslutaBild,  x: 1830,  y: 0,  observer: this);
        }
```

Bild 4. Koden avgör vilken av bilderna *villAvslutaBild* och *avslutaBild* som ska visas.

2.4.3 Svårighetsgrader och strukturförändringar

För att ge variation till spelet skapades olika svårighetsgrader, där en högre nivå av svårighetsgrad innebär att mer information söks i spelet, vilket gör det svårare. Dessa svårighetsgrader var *Lätt*, *Svår* samt *Mycket svår*. För den lättaste svårighetsgraden visades vid det här laget av spelets utveckling en silhuettbild av landet, samt landets flagga komplett med färger. Det spelaren skulle behöva gissa rätt på för varje land för att kunna gå vidare till nästa land var endast landets namn och huvudstad. Skulle spelaren istället välja svårighetsgraden "Svår" visades inte flaggan, utan spelaren behövde istället nämna vilken typ av flagga som landets flagga var av (t.ex. kors för Sverige), och behövde sedan gissa flaggans alla färger, en färg i taget, i den ordning som angavs på den bild som visades. Denna funktion var inte färdigkodad ännu, men spelets dåvarande kod anpassades för att senare kunna fullständigt implementera funktionen. Se Bild 5 för ett exempel på hur Sveriges flagga stegvis färgas med funktionen.

Bild 5. De flaggbilder som tillhör landet Sverige i spelet.

Svårighetsgraden *Mycket svår* var tänkt att inkludera en funktion där tre konturbilder av olika flaggor skulle visas efter att korrekt flaggtyp gissats. Spelaren skulle då behöva gissa vilken av de tre bilderna som tillhör det land som visas. Denna funktion kodades inte fullständigt på en gång, då spelet endast innehöll två stycken länder och då funktionen skulle kräva tre

stycken (eftersom tre stycken bilder skulle visas). Ett lands silhuettbild skulle för denna svårighetsgrad visas i ett roterat läge, då för att göra det svårare för spelaren att lista ut vilket land som visas. För att rotera bilden användes en modifierad version av den kod som användare Reverend Gonzo publicerade i följande Stackoverflow-tråd: https://stackoverflow.com/questions/4156518/rotate-an-image-in-java/4156760#4156760. Bild 6 och Bild 7 visar den modifierade koden med kommentarer.

```
//Vinkeln slumpas här för att den inte ska uppdateras varje gång spelet uppdaterar det som visas.
// 2 pi radianer = 360 grader.
silhuettRot = Math.random() * 2 * Math.PI;
```

Bild 6. Silhuettbildens rotation slumpgenereras.

```
//Får ut absolutbeloppet för sinus och cosinus för den tidigare slumpade vinkeln.
// Ger alltså alltid ett positivt värde.
double sin = Math.abs(Math.sin(silhuettRot)), cos = Math.abs(Math.cos(silhuettRot));

//Skapar variabler med värden för bildens höjd/bredd före och efter rotationen
double bredd = land.silhuett.getWidth(), hojd = land.silhuett.getHeight();
int nyBredd = (int) Math.floor(bredd * cos + hojd * sin), nyHojd = (int) Math.floor(hojd * cos + bredd * sin);

//Skapar bilden som ska roteras (bi) och en Graphics2D (utöver "g" som redan finns).
//Allt som denna Graphics2D kommer att rita (bilden "bi") kommer att roteras.
//Det behövs en separat Graphics2D (utöver "g") för att inte rotera ALLT som ska visas.
BufferedImage bi = GraphicsEnvironment.getLocalGraphicsEnvironment().getDefaultScreenDevice()
        .getDefaultConfiguration().createCompatibleImage(nyBredd, nyHojd, Transparency.TRANSLUCENT);
Graphics2D g3 = bi.createGraphics();

//Ser till så att bildens "bounds", d.v.s. upptagningsområde flyttas till rätt ställe efter rotation.
g3.translate( tx: (nyBredd - bredd) / 2,  ty: (nyHojd - hojd) / 2);

//Roterar bilden
g3.rotate(silhuettRot,  x: bredd / 2,  y: hojd / 2);
g3.drawRenderedImage(land.silhuett,  xform: null);
g3.dispose();

//Ritar den roterate bilden
g.drawImage(bi,  x: 800,  y: 250,  observer: this);
```

Bild 7. Kod som behandlar silhuettbildens rotation.

Efter svårighetsgraderna och relaterade funktioner hade skapats ändrades följande fält och strukturer:

- BufferedImage används nu för bilder istället för ImageIcon eftersom en BufferedImage är mer lättmanipulerad, vilket behövs för att kunna rotera bilder. Egentligen krävdes endast ett klassbyte för silhuettbilderna, men då skillnaden mellan klasserna ImageIcon och BufferedImage inte påverkade spelets andra funktioner byttes även klassen för resterande bilder. Detta bidrog delvis till en bättre struktur i koden, men även till att samma metoder kunde användas för de olika bilderna.
- Fälten *flaggKont* och *flaggFull* byttes ut mot en int, *antalFarger*, som anger antalet färger i varje lands flagga. Varje land tilldelas nu en array vid initialisering, *flaggArray*, som innehåller landets alla flaggbilder.

- Fältet *världsdel* togs bort och byttes ut mot att initiera varje land samtidigt som det läggs till i en array motsvarande den region som landet tillhör. Bild 8 visar hur ett land lades till i spelet vid det här laget.

```
nordenArray.add(new Land( namn: "Sverige",  huvudstad: "Stockholm", FLAGGTYP_KORS, antalFarger: 2));
```

Bild 8. Landet Sverige läggs till i arrayen nordenArray.

För att få en bättre ordning på spelets bilder finns alla bilder nu i en mapp inuti bilder-mappen med samma namn som det land bilderna tillhör. Bilderna för Norge är exempelvis samlade i mappen *.../bilder/Norge/*. Detta eliminerar behovet av att inkludera landets namn i varje bilds filnamn då varje lands bilder samlas i respektive lands mapp. Silhuettbilderna fick därför namnet *silhuett* och flaggbilderna *flaggaX,* där *X* är antalet färger som visas i flaggan. Norges bildmapp innehåller t.ex. fem stycken bilder: en silhuettbild och fyra stycken flaggbilder (en tom flaggbild (konturbild) samt en flaggbild för varje färg i flaggan). Se Bild 9..

Bild 9. Mappen "Norge" och dess bilder.

För att förbättra spelets struktur och för att göra spelet mer principfast infördes standarder för spelets bilder. Dessa standarder inkluderar att begränsa silhuettbilderna till maximalt 300 pixlar (px) i höjd och bredd, att ge alla flaggbilder höjden 200px, att flaggbilderna ska omges av en svart 1px bred kantlinje samt att typsnittet för flaggbildernas siffror ska vara Impact. Tidigare skapade/infogade bilder korrigerades med Paint.NETs storleks-, kant- och textverktyg för att passa standarden.

2.4.4 Färdigställande av funktioner

För att färdigställa den funktion som låter spelaren gissa en flaggas färger krävs att spelet vet vilka färger som ett lands flagga inkluderar. För att ge stöd åt detta ändrades klassen Lands fält ytterligare en gång, denna gång genom att lägga till fältet *flaggFargerArray* samt genom att byta namn på fältet *flaggArray* till *flaggBilderArray,* för att lättare kunna skilja den från den nyskapade arrayen. Fältet *antalFarger* togs bort då den nu kunde ersättas med metoden *flaggFargerArray.size()*. En flaggas färger anges som en enskild textsträng när ett land initieras, där färgerna åtskiljs med ett blanksteg. Dessa färger läggs sedan till i arrayen *flaggFargerArray* med hjälp av den inbyggda metoden *addAll*. Bild 10 och Bild 11 visar hur länder initieras respektive hur alla färger läggs till i arrayen *flaggFargerArray*.

```
nordenArray.add(new Land( namn: "Sverige",  huvudstad: "Stockholm", FLAGGTYP_KORS,  farger: "blå gul"));
```

Bild 10. Uppdaterad version av samma kodsnutt som i Bild 8.

```
this.flaggFargerArray.addAll(Arrays.asList(farger.split( regex: " ")));
```

Bild 11. Funktion som delar upp en textsträng till mindre textsträngar.

Färggissnings-funktionen färdigställdes genom att låta röststyrningsmetoden jämföra den färg som taligenkänningen uppfattat med den färg som söks. Den färg som söks motsvaras av färg nummer *antalFargerVisade* i arrayen *flaggFargerArray*. Variabeln *antalFargerVisade* är av typen int och har inledningsvis värdet 0 och ökar med 1 varje gång spelaren gissar rätt på en flaggas färg. Då spelaren har gissat rätt på en färg går spelet vidare till nästa färg och bild. Den bild som visas är då bild nummer *antalFargerVisade* i *flaggBilderArray*. Detta motsvarar att visa bilden med namnet "*flaggaX*" i landets bildmapp, där *X* = *antalFargerVisade*. Bild 12 visar den kod ur röststyrningsmetoden som jämför den färg som taligenkänningen uppfattade (*in*) med den färg som söks.

```
//Loopen körs en gång för varje färg i arrayen fargerArray.
//Kör resterande kod om användaren nämnde en färg
for (String fargSomKollas : Konstanter.fargerArray) {
    if (in.contains(fargSomKollas)) {

        //Om användaren sa korrekt färg
        if (fargSomKollas.equals(Spel.land.flaggFargerArray.get(Spel.antalFargerVisade))) {
            Spel.antalFargerVisade += 1;
            System.out.println("Antal färger visade = " + Spel.antalFargerVisade);
            if (Spel.antalFargerVisade == Spel.land.flaggFargerArray.size()) {
                Spel.flaggrev = true;
            }
            korrektSvar( typ: "flaggrev");
            break;
        }
        //Om användaren sa fel färg
        else {

            //Om användaren nämnde en tidigare färg så ges inget felsvar.
            boolean saTidigareFarg = false;
            for (int i = Spel.antalFargerVisade; i > 0; i--) {
                if (fargSomKollas.equals(Spel.land.flaggFargerArray.get(i - 1))) {
                    saTidigareFarg = true;
                    break;
                }
            }
            //Om nämnde en felaktig färg som inte redan har visats
            if (!saTidigareFarg) {
                Spel.antalFargerVisade += 1;
                System.out.println("Antal färger visade = " + Spel.antalFargerVisade);
                if (Spel.antalFargerVisade == Spel.land.flaggFargerArray.size()) {
                    Spel.flaggrev = true;
                }
                //Ger felsvar
                switch (fargSomKollas) {
                    default:
                        felsvar( s: "Felaktig färg. Du sa " + fargSomKollas + ". Korrekt svar: "
                                + Spel.land.flaggFargerArray.get(Spel.antalFargerVisade - 1));
```

Bild 12. Den kod som kontrollerar uppfattad färg.

Med färggissnings-funktionen klar färdigställdes den funktion som låter spelaren gissa korrekt konturbild för ett lands flagga. Denna funktion är i grunden mycket simpel, eftersom den fungerar genom att visa den första bilden från landets *flaggBilderArray* tillsammans med samma sorts bild från två andra länder. Programmet behöver då endast jämföra den bild som spelaren har gissat på med den som söks. Om bilderna är identiska blir gissningen korrekt, annars inkorrekt.

2.4.5 Buggfixar och förbättringar

Efter svårighetsgradernas funktioner hade kodats färdigt testades funktionerna. Det uppmärksammades då en bugg, det vill säga ett kodningsfel, som gjorde att spelaren automatiskt fick ett felsvar när programmet gick från ett land till ett annat. Problemet visade sig vara att röststyrningsmetoden kallades med information från föregående land varje gång ett nytt land visade. Om spelet gick från Sverige till Norge "gissade" programmet direkt med det som spelaren hade sagt om Sverige. Orsaken visade sig vara att GCS har svårt att uppfatta när en mening avslutas om det inte ges en längre paus i tal. Allt som spelaren hade sagt adderades till en enda lång mening, så när röststyrningsmetoden kallades med meningen fanns det ibland information för andra länder än det som visades, vilket då gav ett felsvar.

Problemet löstes inledningsvis genom att skapa en textsträng, *detSomSadesTidigare*, som vid varje taluppfattning av GCS tilldelades ett värde motsvarande den mening som GCS uppfattade. Innan den uppfattade meningen går till röststyrningsmetoden subtraherades antalet tecken i *detSomSadesTidigare* från meningen, detta med hjälp av de inbyggda metoderna *String.substring(X)* och *String.length()*. Den förstnämnda metoden tar bort *X* stycken tecken från den textsträng som metoden kallas hos; den andra metoden ger ett värde motsvarande textsträngens teckenlängd. Lösningen medförde att endast nyligen uppfattad information passerades till metoden, då tidigare information togs bort.

Lösningen fungerade i viss mån, men istället för att ge felsvar kraschade spelet ibland när ett nytt land visades. Orsaken till dessa kraschar var att när GCS uppfattade en ny mening försökte metoden *String.substring(X)* ta bort tecken från en textsträng som ibland innehöll färre tecken än *detSomSadesTidigare.length()*. Om metoden försökte ta bort fler tecken än vad som fanns tillgängligt kraschade programmet.

Problemet löstes genom att skapa två booleans, *nyssKorrekt* samt *nyssFelsvar*, vilka tilldelas värdet *true* vid ett korrekt svar respektive ett felaktigt svar, samt värdet *false* då GCS uppfattar en ny mening. Dessa kombinerades tillsammans med ett villkor som ser till att *detSomSadesTidigare.length()* är mindre än den textsträng som tecken subtraheras från, då för att garantera att endast rätt innehåll borttas samt för att se till att spelaren inte får repetitiva felsvar. Se Bild 13.

```
if (nyssKorrekt) {
    if (!nyssFelsvar) {
        // "res" = det tal som taligenkänningen (GCS) har uppfattat.
        // "d" = det som sagts EFTER att användaren fått rätt.
        if (res.length() <= detSomSadesTidigare.length()) {
        } else if (res.length() > 2 && detSomSadesTidigare.length() > 2) {
            String d = res.substring(detSomSadesTidigare.length() - 1);
            Logik.check(d);
        } else {
            Logik.check(res);
        }
    }
} else {
    if (res.length() > 0) {
        detSomSadesTidigare = res;
    }
    if (!nyssFelsvar) {
        Logik.check(res);
    }
}

// "result.getIsFinal()" är en boolean från GCS som har värdet
// "true" då GCS har uppfattat en mening som avslutad.
if (result.getIsFinal()) {
    nyssFelsvar = false;
    nyssKorrekt = false;
    detSomSadesTidigare = "";
```

Bild 13. Kod som ser till att korrekt textsträng kontrolleras.

Spelet förbättrades ytterligare genom att vid felsvar visa spelaren vad den svarat fel på, detta genom att kräva en textsträng som parameter när funktionen *felsvar()* kallas. Denna textsträngs innehåll ritas sedan upp på skärmen när metoden *Rita.repaint()* kallas, och finns kvar tills spelaren gör en korrekt gissning.

```
felsvar( s: "Fel namn på landet. Du sa '" + denSomSades + "' Rätt svar: " + Spel.land.namn);
```

Bild 14. Funktionen *felsvar()* kallas då användaren gissar fel på ett lands namn.

Spelets liv-system togs bort och ersattes med ett poängsystem som anger hur stor andel av spelarens gissningar som är korrekta. Detta tillåter spelaren att alltid spela färdigt spelet, samt fungerar bättre i ett highscore-system som vid det här laget planerades att läggas till i spelet.

Ytterligare en förbättring som implementerades var att skapa ett filter som ersätter vissa ord som taligenkänningen felaktigt uppfattar med de ord som söks. Det som filtret främst gör är att korrigera felstavningar samt att korrigera versaler. Bild 15 visar hur det av GCS

uppfattade “Lichtenstein” korrigeras till “Liechtenstein”.

```
if (res.contains("Lichtenstein")) {
    res = res.replaceAll( regex: "Lichtenstein",  replacement: "Liechtenstein");
}
```

Bild 15. Exempel på korrigering av taligenkänningens missuppfattningar.

Spelupplevelsen försökte förbättras genom att lägga till ljudeffekter vid korrekta och inkorrekta gissningar samt genom att för spelaren visa den text som taligenkänningen uppfattar. Dessa funktioner togs dock bort under utveckling då de inte tycktes bidra med något positivt till spelet. Ljudeffekterna kunde vara i vägen för taligenkänningen, och att visa allt som taligenkänningen uppfattar ansågs vara för distraherande.

2.4.6 Spelets innehåll utökas

I den senare delen av arbetets gång spenderades omkring tio timmar till att lägga till samtliga europeiska länder i spelet, detta genom att hämta/skapa lämpliga silhuett- och flaggbilder och att sedan initiera ett objekt av klassen *Land* med parametrar som motsvarar landets egenskaper. Objektet initieras samtidigt som det läggs till i en array som motsvarar den del av Europa som landet tillhör (t.ex. landet Vitryssland till arrayen Östeuropa). För att bestämma vilken del av Europa som ett land tillhör användes FN:s definition.[6] Arrayen *nordenArray* togs bort och ersattes med *nordArray,* som motsvarar Nordeuropa.

Tillsammans med svårighetsgraderna behöver spelaren nu i spelets huvudmeny även välja vilken del av Europas länder som spelet ska innehålla. Istället för att behöva ta sig förbi samtliga 44 länder i spelet visas nu endast nio till femton länder beroende på vilken del av Europa som spelaren valt. När alla länder är tillagda i respektive regions array läggs tio stycken slumpvis valda länder till i en array med namnet *blandadArray*, vilken låter spelaren spela med tio stycken länder utan förvald region.

Då antalet länder och huvudstäder blev fler ökade som följd även antalet länder och huvudstäder som kunde ge ett felsvar, eftersom koden som gav felsvar för dessa delar jämförde landet/huvudstaden som uppfattades med spelets resterande länder/städer. För att ge stöd åt fler möjliga felaktiga svar utan att lägga till fler länder i spelet lades namnet på världens alla länder, huvudstäder och största städer ihop i två olika textdokument: ett för länder samt ett för städer. Istället för att jämföra med spelets resterande länder/städer jämför spelet nu uppfattad stad/land med respektive listas innehåll. Detta leder till att en spelare exempelvis kan få ett felaktigt svar för att gissa att Tysklands huvudstad är Tokyo, Japans huvudstad, trots att landet Japan inte finns med i spelet. Länderna och städerna hämtades från tabeller från Wikipedia.[7] [8] [9]

---

[6] https://en.wikipedia.org/wiki/United_Nations_geoscheme_for_Europe (Hämtad 2019-11-15)

[7] https://sv.wikipedia.org/wiki/Lista_%C3%B6ver_Europas_st%C3%B6rsta_st%C3%A4der (Hämtad 15/11/2019)

[8] https://sv.wikipedia.org/wiki/Lista_%C3%B6ver_sj%C3%A4lvst%C3%A4ndiga_stater (Hämtad 15/11/2019)

[9] https://sv.wikipedia.org/wiki/Lista_%C3%B6ver_huvudst%C3%A4der (Hämtad 15/11/2019)

2.4.7 Highscore-system och reviderade svårighetsgrader

Svårighetsgraderna reviderades då dessa ansågs kunna förbättras genom att lägga till ytterligare en svårighetsgrad samt genom att bättre bestämma hur mycket information som ska krävas av spelaren för respektive svårighetsgrad. Att spelet nu inkluderar fyra stycken svårighetsgrader istället för tre bidrar till att spelaren får en större valmöjlighet över hur utmanande spelet ska vara. Svårighetsgraderna är nu följande:

- *Enkelt:*
  - Information given för varje land:
    - Silhuettbild
    - Huvudstad
    - Fullständig flagga
  - Information som söks för varje land:
    - Namn
- *Medelsvårt:*
  - Information given för varje land:
    - Silhuettbild
    - Tre olika fullständiga flaggor, varav en är korrekt för landet och två är från två andra slumpmässigt utvalda länder
  - Information som söks för varje land:
    - Namn
    - Huvudstad
    - Korrekt flagga av de tre som visas
- *Svårt:*
  - Information given för varje land:
    - Silhuettbild
    - Tre flaggors konturbilder, varav en är korrekt för landet och två är från två andra slumpmässigt utvalda länder med samma flaggtyp
  - Information som söks för varje land:
    - Namn
    - Huvudstad
    - Korrekt konturbild av de tre som visas
    - Flaggans färger
- *Överdrivet svårt*
  - Information given för varje land:
    - Slumpmässigt roterad silhuettbild
    - Då spelaren har gissat korrekt flaggtyp visas tre flaggors konturbilder likt som för *Svårt*
  - Information som söks för varje land:
    - Namn
    - Huvudstad
    - Flaggtyp
    - Korrekt konturbild av de tre som visas
    - Flaggans färger

Med svårighetsgraderna reviderade skapades ett highscore-system som låter spelaren spara sitt resultat, detta genom att bevara resultatet i ett textdokument. En spelares resultat innefattar spelarens andel korrekta gissningar, spelarens namn samt tiden det tog att spela

färdigt spelet. Dessa värden lagras i ett textdokument som motsvarar den region och svårighetsgrad som spelaren valt. Det bästa resultatet för varje kombination av region och svårighetsgrad visas i en tabell i spelets huvudmeny, där det bästa resultatet är det med störst andel korrekta gissningar. Om två spelare har samma andel korrekta gissningar blir highscore resultatet från den av spelarna som klarade spelet på den kortaste tiden. Tiden sparas med en millisekunds noggrannhet med hjälp av funktionerna som Bild 16 visar.

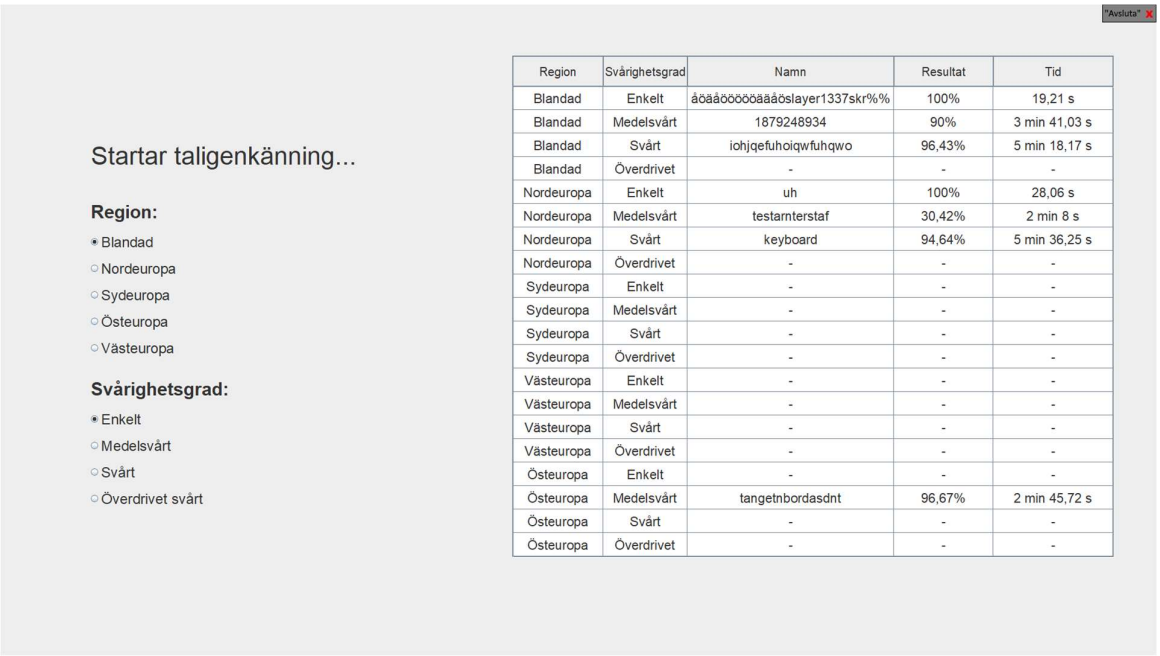
```
private static long start;

static void startaTiden() {
    start = System.currentTimeMillis();
}

static double tidSomPasserat() {
    long nu = System.currentTimeMillis();
    return (nu - start) / 1000.0;
}
```

Bild 16. Speltiden beräknas.

Med spelets alla funktioner färdiggjorda samt med alla europeiska länder tillagda ansågs spelet vara färdigt. Bild 17 och Bild 18 visar hur spelet ser ut i huvudmenyn, respektive vid svårighetsgraden *Medelsvårt*.



Region	Svårighetsgrad	Namn	Resultat	Tid
Blandad	Enkelt	åöääöööööääöslayer1337skr%%	100%	19,21 s
Blandad	Medelsvårt	1879248934	90%	3 min 41,03 s
Blandad	Svårt	iohjqefuhoiqwfuhqwo	96,43%	5 min 18,17 s
Blandad	Överdrivet	-	-	-
Nordeuropa	Enkelt	uh	100%	28,06 s
Nordeuropa	Medelsvårt	testarnterstaf	30,42%	2 min 8 s
Nordeuropa	Svårt	keyboard	94,64%	5 min 36,25 s
Nordeuropa	Överdrivet	-	-	-
Sydeuropa	Enkelt	-	-	-
Sydeuropa	Medelsvårt	-	-	-
Sydeuropa	Svårt	-	-	-
Sydeuropa	Överdrivet	-	-	-
Västeuropa	Enkelt	-	-	-
Västeuropa	Medelsvårt	-	-	-
Västeuropa	Svårt	-	-	-
Västeuropa	Överdrivet	-	-	-
Österuropa	Enkelt	-	-	-
Österuropa	Medelsvårt	tangetnbordasdnt	96,67%	2 min 45,72 s
Österuropa	Svårt	-	-	-
Österuropa	Överdrivet	-	-	-

Bild 17. Spelets huvudmeny.

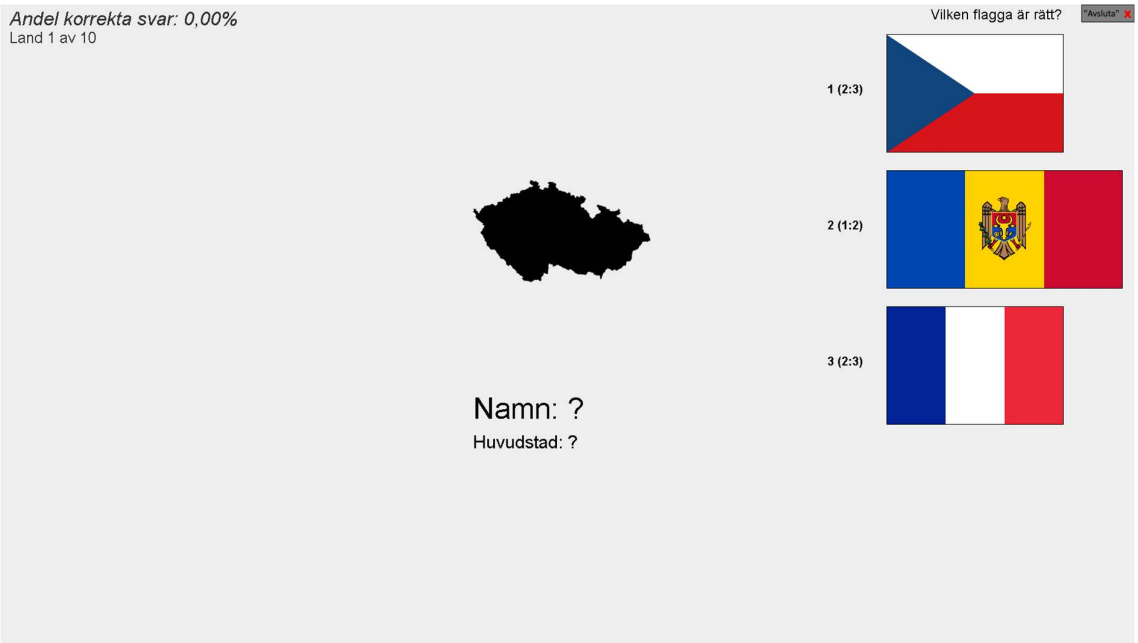


Bild 18. Landet Tjeckien i spelet med svårighetsgraden *Medelsvårt.*

2.5 Redovisning/presentation

Arbetet är tänkt att redovisas i ett tyst rum där besökare får möjligheten att spela spelet. Spelet är tänkt att köras på en dator med tillkopplad skärm, mikrofon och tangentbord. Datorn kommer användas till att köra själva spelet, skärmen till att visa dess innehåll, mikrofonen till att ta upp spelarens röst och tangentbordet till att låta spelaren skriva in sitt namn om den vill spara sitt resultat.

3. Resultat av projektet

Spelet utvecklades fullständigt och innehåller i sitt färdiga skick alla Europas länder, vilket var ett ambitiöst mål som uppnåddes. Spelet innehåller de planerade funktionerna och ytterligare funktioner vilka inte var planerade från början, men som förbättrar spelupplevelsen. Till exempel highscore-systemet.

Spelet uppfyller målet att kunna kontrolleras fullständigt med endast rösten som inmatning samt även målet med att vara lärorikt. Om en spelare gissar felaktigt visas ändå korrekt information, vilket bidrar till att spelaren lär sig vilket svar som är korrekt.

Arbetet har även uppfyllt sitt syfte, vilket var att ge en bättre förståelse över taligenkänning och röststyrning, samt att förbättra kunskaper inom programmering. Arbetet har bidragit med utmaningar inom dessa områden där lösningar både har hittats och skapats, vilket har lett till en fördjupad förståelse inom områdena.

Frågeställningarna har besvarats tack vare arbetet. Den första frågeställningen handlade om hur taligenkänning kan implementeras i Javaprogram, vilket svaret för detta spel var den lyckade implementationen av Google Cloud Speech-to-Text. Den andra frågeställningen handlade om hur röststyrning kan skapas med hjälp av taligenkänning. Röststyrning skapades i projektet genom att skapa en metod i Java som jämför det som uppfattats av taligenkänningen med den information som söks.

4. Värdering

4.1. Värdering av resultatet

Det färdiga spelet inkluderar alla planerade funktioner samt alla Europas länder, och utvecklades till ett fullständigt och komplett skick. Det planerades att spelet skulle inkludera ett sådant stort antal av Europas länder som möjligt inom arbetets tidsperiod; att inkludera samtliga europeiska länder ansågs vara ett ambitiöst mål som endast skulle uppnås om det fanns mycket tid kvar till arbetet efter att spelets funktioner hade färdigställts.

Det enda negativa med resultatet är hur programmeringskoden är uppbyggd. Strukturen kan ses som någorlunda slarvig, och repetitiv kod förekommer, vilket beror på brist av kunskaper och erfarenheter gällande programmering av större applikationer. Spelet fungerar dock som det ska; det enda som påverkas är hur lätt koden kan förstås av någon som läser den samt hur bra spelet presterar, men prestandan är tillräckligt bra för att spelet ska kunna spelas störningsfritt.

Resultatet blev sammanfattningsvis mycket bra. Arbetet var under hela projektets gång i fas enligt Gantt-schemat, och ingen del av schemat behövde revideras då det fanns nog med tid för att hinna med varje del av arbetet.

4.2. Värdering av den egna insatsen

Den egna insatsen var under detta arbete mycket bra, då det arbetades under varje arbetspass samt även på egen fritid. Arbetet sågs alltid till att vara i fas enligt Gantt-schemat, och om mer tid behövdes för ett särskilt moment utfördes arbete på egen tid.

5. Källförteckning

Elektroniska källor

Reverend Gonzo. Stack Overflow. *Rotate an image in java*. Publicerad 11/11/2010 https://stackoverflow.com/questions/4156518/rotate-an-image-in-java/4156760#4156760 (Hämtad 15/10/2019)

Skolverket. *Teknikprogrammet*. https://www.skolverket.se/undervisning/gymnasieskolan/laroplan-program-och-amnen-i-gymnasieskolan/gymnasieprogrammen/program?url=1530314731%2Fsyllabuscw%2Fjsp%2Fprogram.htm%3FprogramCode%3DTE001%26tos%3Dgy&sv.url=12.5dfee44715d35a5cdfa9295#anchor_1 (Hämtad 2019-09-04)

Wikipedia. *Lista över Europas största städer.* https://sv.wikipedia.org/wiki/Lista_%C3%B6ver_Europas_st%C3%B6rsta_st%C3%A4der (Hämtad 15/11/2019)

Wikipedia. *Lista över huvudstäder.* https://sv.wikipedia.org/wiki/Lista_%C3%B6ver_huvudst%C3%A4der (Hämtad 15/11/2019)

Wikipedia. *Lista över självständiga stater.* https://sv.wikipedia.org/wiki/Lista_%C3%B6ver_sj%C3%A4lvst%C3%A4ndiga_stater (Hämtad 15/11/2019)

Wikipedia. *Lista över världens största storstadsområden.* https://sv.wikipedia.org/wiki/Lista_%C3%B6ver_v%C3%A4rldens_st%C3%B6rsta_storstadsomr%C3%A5den (Hämtad 15/11/2019)

Wikipedia. *United Nations geoscheme for Europe*. https://en.wikipedia.org/wiki/United_Nations_geoscheme_for_Europe (Hämtad 15/11/2019)

Bilaga 1

Boolean: En datatyp som antingen har värdet *true* (sann) eller *false* (falsk).

Funktion: En del av ett datorprogram som kan anropas för att utföra en särskild uppgift. Funktioner används för att undvika repetitiv kod.

Fält: Data som tillhör en specifik klass eller ett objekt.

Int: En datatyp som omfattar positiva och negativa heltal.

Klass: Ett avsnitt programkod som omfattar relaterade attributer och metoder. Fungerar som en mall för objekt.

Metod: En funktion som tillhör en klass eller en instans.

Objekt: En instans av en klass.

Parameter: Värde som anges då en metod kallas i Java. Värdet förs vidare till den kallade metoden.

Silhuett: Enfärgad bild som enbart består av en kontur.

String: En datatyp som omfattar text.