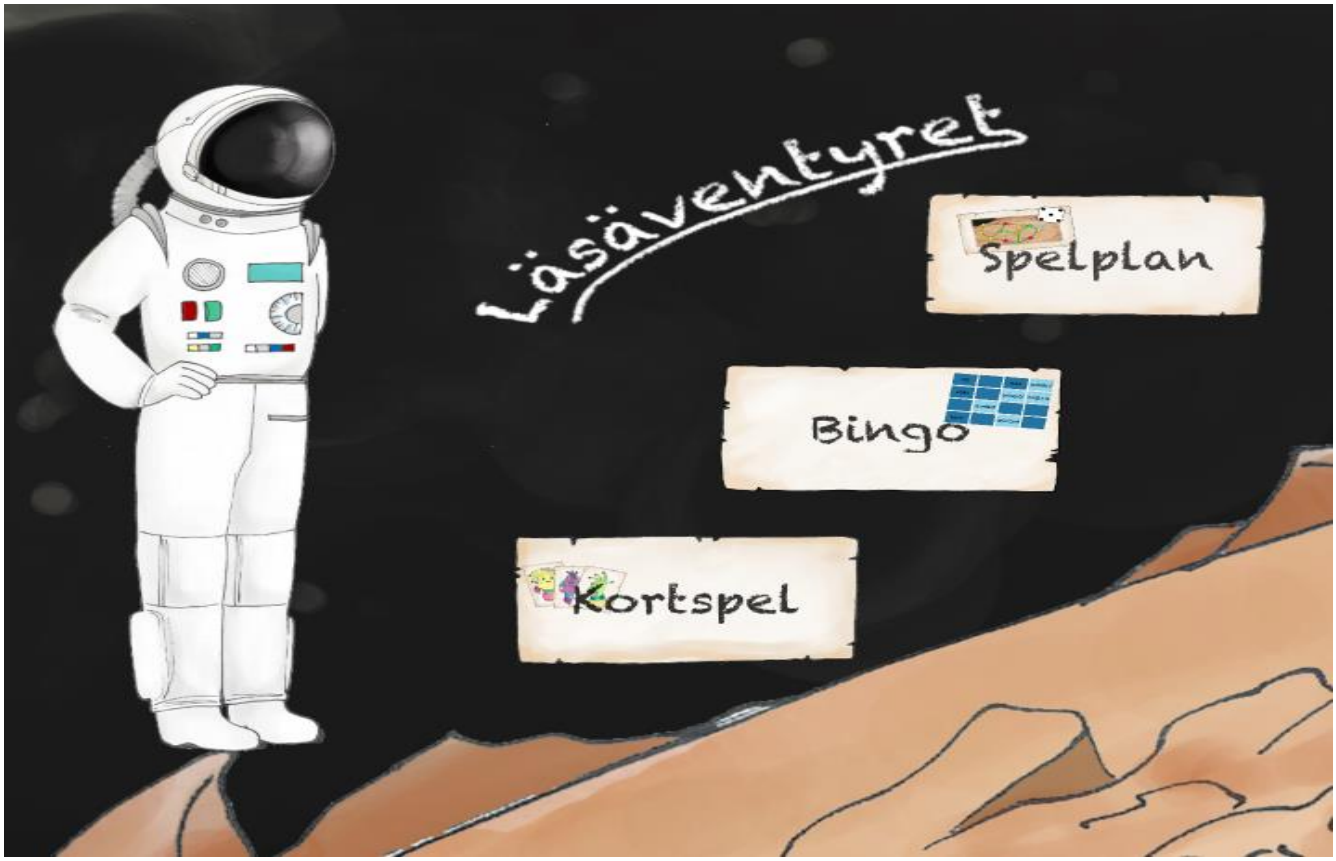
CHALMERS



Utveckling av läsförståelse-applikation för barn 6 till 9 år

Bachelor's thesis in Computer Science and Engineering

Benjamin Wijk
Christoffer Krull

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

DEGREE PROJECT REPORT

Utveckling av pedagogisk läsförståelseapplikation för iPhone och iPad

Benjamin Wijk
Christoffer Krull

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Utveckling av pedagogisk läsförståelseapplikation för iPhone och iPad
Benjamin Wijk
Christoffer Krull

Examiner:
Christer Carlsson

Department of Computer Science and Engineering
Chalmers University of Technology/University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
A screenshot showing the main menu of the developed application “Läsäventyret”

Department of Computer Science and Engineering
Gothenburg 2017

ABSTRACT

The project was put together by Lena Levin to create a reading comprehension application that reflects the work she does as a private tutor. We have created an application that uses the same principles and pedagogy that Lena used during her time as a teacher. The work was carried out at Chalmers Lindholmen in Gothenburg Sweden. The application meets the employer's specifications and uses words that sound the way they are spelled to teach students between 6 and 9 years old about compound words. The application contains three different exercises to teach the content in a varied way. The programming language Swift was used to develop the application for the iOS platform. The application can be used offline and has no limits on where and when it can be used.

Keywords: iOS, Application, Reading Comprehension

SAMMANFATTNING

Projektet startades av uppdragsgivaren Lena Levin för att skapa en applikation för att lära ut läsförståelse vilket reflekterade det arbete hon gör med sina elever som privatlärare. Det fanns ambitioner att skapa en applikation som utgår från samma principer och pedagogik hon utvecklat under sin tid som lärare. Arbetet utfördes på Chalmers Lindholmen i Göteborg. En applikation har skapats efter arbetsgivarens instruktioner som använder sig av ljudenliga ord för att lära ut sammansatta ord. Applikationen innehåller tre olika övningar för att lära ut innehållet på ett varierande sätt. Dessa övningar innefattar ett kortspel, ett bingospel och en spelplan. Programmeringsspråket Swift användes för att utveckla applikationen till plattformen iOS. Applikationen är utvecklad endast för iOS och fokuserar på att användas av barn i målgruppen 6 till 9 år.

Nyckelord: iOS, Applikation, Läsförståelse

PREFACE

Examensarbetet genomfördes på Chalmers Lindholmen och Johanneberg och båda projektmedlemmarna går sitt sista år på den treåriga Datatekniklinjen för högskoleingenjörer. Examensarbetet omfattar tre år av studier och använder sig av principer och tekniker vi har lärt oss på vår tid i programmet, speciellt fokus på utveckling av applikationer och studier inom arbetsflöde och kodning. Vi vill ge ett stort tack till Peter Lundin som hjälpte oss komma igång med projektet och satte oss i kontakt med Lena Levin. Vi vill även ge ett tack till Sakib Sistek som hjälpte oss med hårdvaran i form av Mac-datorer vilket behövdes för att utveckla applikationen. Ett tack till Alex Gerdes vår handledare i examensarbetet som hjälpte oss med planering och allmänna frågor kring projektet. Sist av allt vill vi ge ett jättestort tack till Lena Levin som valde att genomföra detta projektet med oss, vi vill tacka henne för det hon har lärt oss om pedagogik inom applikationsutveckling och för den tid hon har lagt ner på detta projekt.

Table of Contents

Terminologi och Förkortningar

Frågeord	Ordet som användaren ska gissa på.
Gesture Recognizer (GR)	En klass som hanterar interaktion med touchscreens.
Svarsalternativ	Refererar till alternativ som man kan matcha med frågeordet.
Korthand	Samma som svarsalternativ.
MVC	Står för Model-View-Controller och är ett arkitekturmönster för att strukturera koden.
Scene	Visuell representation av en ViewController.
View	En grundläggande klass för alla visuella objekt på skärmen.
ViewController (VC)	En term inom Swift som beskriver en klass som manipulerar en view.
Storyboard	En visuell representation av en ViewController, används för att designa mer detaljerade UI element.
Swift	Ett programmeringsspråk utvecklat av Apple.

1. Inledning

1.1 Bakgrund

Den kraftiga ökningen av mobila enheter gör det idag möjligt för utvecklare att nå ut till en stor mängd användare med sina ideér. Idag finns det en stor marknad för undervisningsapplikationer för barn och ett antal framgångsrika applikationer på marknaden. Uppdragsgivaren Lena Levin ser flera problem med dessa applikationer, framförallt är flera av dessa inte pedagogiskt designade och presenterar inte övningarna på ett intuitivt sätt för barn i den avsedda målgruppen. Lena vill använda den typen av undervisning hon utnyttjar i sitt dagliga arbete som privatlärare för att skapa ett utbildningsspel för mobila enheter. Denna undervisning fokuserar på att lära ut sammansatta ord med hjälp av ljudenliga ord och är ett bra sätt att lära ut dessa koncept. Lena vill med denna applikation utnyttja pedagogiska principer hon har förfinat under sina år som lärare.

1.2 Syfte

Applikationen har ett grundläggande syfte att göra det möjligt för fler barn att ta del av Lenas undervisningsmetoder. Tanken är att modellera applikationen efter de pedagogiska övningar Lena använder med sina elever när hon lär ut sammansatta ord. Detta gör det möjligt för användaren att lära sig sammansatta ord på egen hand och på så sätt själv kontrollera sin undervisning.

1.3 Mål

Applikationen skall vara avsedd att användas av både pojkar och flickor i målgruppen 6 till 9 år.

Applikationen ska innehålla tre spel vars tanke är att ge användaren möjlighet att välja inlärningsprocess beroende på vad som känns engagerande. Samtliga spel kommer att använda sig av samma ordlista, där den avgörande skillnaden är hur spelet utspelar sig. Ordlistan kommer endast bestå av ljudenliga ord.

Användaren ska ha möjlighet att välja inlärningsnivå beroende på hur avancerat den finner inlärningsmaterialet. Detta gör att användare som har lätt för att lära sig övningarna kan gå vidare till mer avancerat material utan att behöva gå igenom alla nivåer, medan andra användare har möjligheten att fortsätta öva på den nuvarande nivån.

Varje sammansatt ord i applikationen kommer att ha en ljudfil associerad med sig. Denna ljudfil används för att lära användaren hur ordet uttalas.

Applikationen ska också ha ett belöningssystem som känns tillfredsställande för målgruppen. Systemet skall inte vara ett vanligt pokalsystem med guld-, silver- och brons-stjärna. Idén är att skapa ett system där användaren får välja en karaktär som sedan “låser upp” olika uppgraderingar under spelets gång.

Applikationen skall kunna användas överallt utan Internetuppkoppling. Detta gör det möjligt att använda applikationen medan man reser eller är utomlands utan att betala eventuella datakostnader. Det gör även applikationen säkrare då föräldrar slipper oroa sig för att användaren på något sätt besöker Internet på ett okontrollerat sätt.

1.4 Avgränsning

Utveckling av applikationen kommer endast att ske till Appleprodukter. Projektgruppen har valt denna plattform efter uppmaningar från uppdragsgivaren som har observerat att majoriteten av undervisningsapplikationer finns på iOS och har därmed redan en bekräftad användarbas. Valet att endast utveckla för en plattform är på grund av tidsbrist och en applikation till Android i framtiden är fullt genomförbar.

2. Metod

För att skapa en applikation som når upp till målen påbörjas projektet med forskning och inlärning av programmeringsspråket Swift samt utvecklingsmiljön Xcode. Projektet utvärderas därefter utifrån projektbeskrivningen och ett grundläggande användargränssnitt kan skapas.

Projektet delas upp i olika kategorier: ordhantering, meny, kortspel, bingo och spelplan. Efter att det grundläggande användargränssnittet skapats bör ordhanteringen defineras. Ordhanteringen fungerar som grund till de olika övningarna, vilket innebär att det behöver vara fastställt vilka av dess metoder övningarna kommer att kunna använda. Bortsett från ordhanteringen kommer övningarna att vara fristående från varandra och utveckling kan ske i valfri ordning. Möten med uppdragsgivaren sker frekvent för att kunna utvärdera om implementationen kommer att fungera bra i praktiken, eller om kompromisser måste göras.

För att lätt kunna koppla ihop arbetet som gjorts av gruppmedlemmarna används versionshanteringsprogrammet Git. Detta möjliggör även återställning av kod till ett tidigare stadie om något går fel.

Testning utförs på två olika sätt, beroende på vilken del som berörs. För den underliggande logiken används unit tests för att fånga majoriteten av buggar. Det grafiska gränssnittet utnyttjar temporära kodstycken för att kontrollera om kod exekveras korrekt vilket betyder att man skapar temporära variabler som man sedan skriver ut i konsollen.

Ljudfiler för de sammansatta orden skapas av arbetsgivaren och behöver endast importeras till projektet. För att skapa ljudfilerna används webbsidan Acapellabox.se som låter användaren skapa ljudfiler med hjälp av ett avancerat “text to speech”-program.

3. Teknisk Bakgrund

Kapitlet beskriver system och tekniker som används inom projektet.

3.1 Utvecklingsmiljö och språk

Utvecklingsmiljö refererar till programmen som används för att skriva och kompilera kod inom ett projekt. Programmeringsspråk är det språk vilket programmet skrivs i. Varje språk har egna konventioner, fördelar och nackdelar vilket gör dem fördelaktiga i olika situationer.

3.1.1 Xcode

Xcode är en utvecklingsmiljö skapad av Apple för att utveckla applikationer för iOS. Versionen av Xcode som användes inom projektet är 7.3.1 och kördes på OS X El Capitan. Inom projektet används Xcode för att utveckla en applikation med programmeringsspråket Swift, men Xcode stödjer även andra programeringsspråk [1].

Xcode har flera olika hjälpmedel för att underlätta utvecklandet av applikationer. En av dessa är Storyboards, vilket används för att designa de visuella delarna av applikationen. Storyboards ger utvecklaren möjlighet att få en överblick av hur applikationen kommer att se ut, utan att behöva köra applikationen. Detta är en visuell representation av en `ViewController`, vilket används inom Xcode för att kontrollera visuella element på skärmen. Storyboards byggs upp av ett antal `Scenes` som innehåller `Views`.

Scenes är en visuell representation av en klass inom applikationen och när användaren går från ett spel till ett annat så byter applikationen Scene. Xcode har även stöd för simulatorer, vilket gör det möjligt att simulera alla former av iOS-produkter.

3.1.2 MVC

MVC står för Model-View-Controller och är en programarkitektur med syfte att göra kodstrukturen mer lätthanterlig och potentiellt återanvändbar. Ofta handlar det om separation av visuella element och data. Detta realiseras genom att kategorisera och separera koden in i tre roller: Model, View och Controller.

Model hanterar all väsentlig data vilket skall användas, exempelvis nästa ord, nuvarande nivå och spelarens karaktärsval. Model sköter även den logiska strukturen vilket manipulerar datan. Model bör vara funktionell utan kunskap om hur det grafiska gränsnittet ser ut. View tar hand om all visuell data och hur saker representeras grafiskt. Om data från Model utnyttjas så bestämmer View-delen hur denna visas upp. Controller sköter kommunikationen mellan Model och View. Ofta behöver View känna till Model för att kunna hämta relevant data, vilket Controller hjälper till med. Controller styr även eventuella knapptryck från användare och skickar dessa vidare till relevanta delar i Model och View.

I Xcode är det vanligt att man kombinerar View och Controller. Då de allra flesta View-objekt i applikationer har ett antal interaktioner som är unika väljer man istället att låta dess `ViewController` styra dem själv [2].

3.1.3 Swift

Swift är gjort för att utveckla applikationer till iOS-system och utvecklingsmiljön är Xcode. Swift har hämtat inspiration från Objective-C och har en syntax som är lik den som används inom Java. Apple vill med detta uppnå ett språk som är säkert, snabbt och har många olika utvecklingsmöjligheter [3].

Dessa utvecklingsmöjligheter involverar ett antal olika konventioner vilket underlättar för utvecklare att arbeta med språket. För att visuellt representera data på skärmen används `UIView`-klassen inom Swift. En view är en behållare för de visuella element som visas, till exempel bilder och text. Det är vanligt att en view i sin tur innehåller views, som då kallas subviews. De används för att strukturera och gruppera olika element för att underlätta hantering av dessa. Ett standardbibliotek vid namn `UIKit` används av alla visuella delar i applikationen och biblioteket har tillgång till ett antal specialiserade views, exempelvis knappar och bilder som representeras med hjälp av `UIButton` och `UIImageView`.

`ViewController` är en klass inom Swift och används för att koppla ihop de visuella elementen i koden till den bakomliggande modellen. Klassen används delvis för att presentera ny visuell data till användaren, men även för att vidarebefordra information från användaren till modellen. Varje storyboard Scene är kopplad till en specifik `ViewController`.

För att lagra data under spelets gång, till exempel ordlistor, används Dictionarys. Dictionarys är en form av avbildning vilket använder sig av en nyckel och ett värde för att förvara sin data. Varje nyckel eller “key” är associerad med ett specifikt värde eller “value”. Både key och value kan vara godtyckliga värden eller till och med samlingar, till exempel en avbildning.

4. Resultat

4.1 Pedagogik

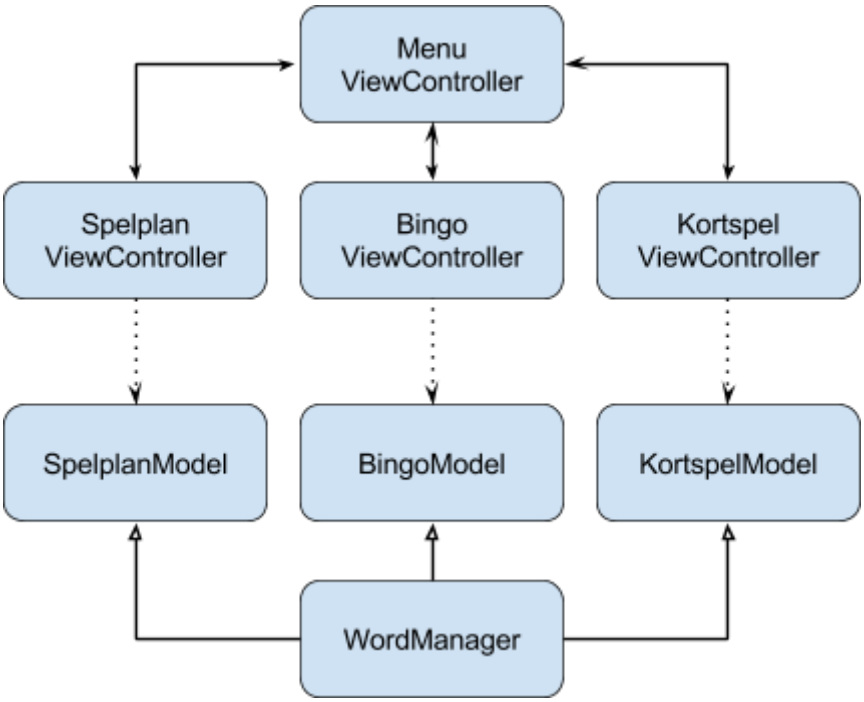
Den pedagogiska tanken bakom applikationen grundar sig i de designval som uppdragsgivaren har specificerat. Applikationen inriktar sig på att ge användaren möjlighet att göra val som påverkar inlärningsprocessen. Anledningen till detta är att uppdragsgivaren har uppmärksammat att elever känner sig mer motiverade att lära sig när de själva får påverka sin undervisning. Valen vilket eleven gör behöver inte vara omfattande utan det är processen av att själv kunna påverka sin miljö som bidrar mest. För att uppnå detta finns det små val vilket användaren får göra för att uppfylla dessa mål. Saker som att kunna välja sin karaktär och att bygga sin egen spelplan bidrar till en känsla av valbarhet.

Andra pedagogiska beslut som gjordes inkluderar val av färger och typsnitt. För att öka läsbarheten för användare med lässvårigheter användes ett lättförståeligt typsnitt genom hela applikationen. Val av färger gjordes för att ge kontrast till olika intressezoner som till exempel knappar och text.

Applikationen centrerar kring att lära ut sammansatta ord med hjälp av ljudenliga ord. Anledningen till detta är för att öka förståelsen för hur sammansatta ord fungerar. Att sätta ihop ord som hus och båt är väldigt enkelt när orden är ljudenliga men att sätta ihop ord som mat och ros är mycket svårare då det sammansatta ordet inte längre uttalas på samma sätt. För att förenkla denna process så finns endast ljudenliga sammansatta ord med i applikationen. Applikationen lär ut sammansatta ord på en grundläggande nivå och går inte in på svårare koncept så som “interfix” vilket är när man sätter in en bokstav mellan de två orden i ett sammansatt ord, till exempel dagstidning.

4.2 Modellering

Projektet delar upp sina klasser enligt en MVC-modell, vilket är fördelaktigt när man utvecklar i Xcode som stödjer denna modell väl. Dessa klasser är uppdelade i modell-klasser och `ViewController`. Somliga delar av applikationen använder sig av `Storyboards` för att designa gränssnittet. I kapitel 5 finns en diskussion om fördelar och nackdelar med att använda `Storyboard` och `ViewController` för att designa gränssnitt. UML-diagrammet i figur 4.1 visar hur applikationen är uppbyggd på en grundläggande nivå.



4.1 Ett UML-diagram vilket beskriver uppbyggnaden av applikationen.

`MenuViewController` används som huvudmeny samt navigationsverktyg för att nå de olika delarna i projektet. Övningarnas `ViewController` använder sin respektive modell-klass för att hämta information. Denna information används av `ViewController` för att manipulera spelet. Samtliga modell-klasser ärver funktioner från huvudmodellklassen `WordManager`. Eftersom de olika övningarna använder samma grundläggande spelregler var detta en smidig lösning för att minimera koduppprepning.

4.3 Utveckling

Applikationen har 6-9-åringar som målgrupp och mycket tid har dedikerats till att göra applikationen anpassad till barn i denna åldern. Det finns inga långa textstycken någonstans i applikationen, spelen skall snabbt kunna förstås även utan tidigare kunskap om hur de fungerar. En till två provomgångar skall räcka innan användaren besitter tillräckligt mycket kunskap för att kunna spela spelet utan större funderingar. Interaktiva element i de olika spelen ger respons på olika sätt för att förmedla vad som sker. Respons sker visuellt, auditivt eller en kombination av båda. Detta gör det möjligt att dra användarens fokus till en speciell händelse på ett naturligt sätt.

4.3.1 WordManager

`WordManager` är den bakomliggande modellklassen alla övningar har som grund. Klassen läser in och lagrar alla frågeord och svarsalternativ samt korrekta matchningar mellan dessa.

Alla övningar använder denna klass för att få tillgång till väsentlig information som till exempel ordlistan, antalet svar och facit. `WordManager` innehåller metoder för att få en ny uppsättning frågeord och svarsalternativ, utvärdera om ett svar är korrekt, utvärdera om övningen är avklarad samt för att nollställa eller gå vidare till nästa nivå.

Specifikationen för `WordManager` har ändrats upprepade gånger under utvecklingen för att kunna hantera både ordnade och oordnade listor samt slumpmoment, vilket har lett till en blandning av avbildningar och Dictionarys. Endast ett fåtal ändringar för att skifta mellan de olika specifikationerna som har uppstått. Den aktuella implementationen av `WordManager` har en bestämd ordning på frågeorden och slumpar endast ordningen av svarsalternativen till frågeordet.

Det första som sker vid initiering av `WordManager` är att funktionen `getWords` anropas. Funktionen läser av en textfil där varje rad innehåller ett frågeord, följt av möjliga svarsalternativ samt korrekta svarsalternativ. `getWords` separerar dessa och lagrar datan i arrayer och dictionarys.

Resterande funktioner anropas av de olika övningarna. Flera av dessa funktioner har möjligheten att markera övningen som avklarad. Detta är ett krav för att övningarna skall ha tillräckligt med information för att utföra rätt animationer, men har ingen större påverkan på `WordManager`.

I samtliga fall är den första funktionen som anropas `nextWord`, som bestämmer nästa frågeord. Den hanterar de olika fall som kan uppstå: när övningen inte är avklarad samt när övningen är avklarad. Den hanterar även basfallet då inget frågeord har bestämts tidigare (första anropet av `nextWord`).

```
func nextWord() -> String{
    //Base value
    if(shownWord == ""){                        är frågeordet tomt?
        shownWord = listOrder[0]                    sätt frågeordet till första värdet i frågeordssamlingen
        listOrderIndex = 0                          nollställ indexräknare
        return shownWord                            returnera frågeordet
    }
    else {                                      annars
        listOrderIndex += 1                         öka indexräknare
    }//Still in bounds?
    if(listOrderIndex+1 <= listOrder.count){    pekar indexräknaren fortfarande på ett giltigt värde i frågeordssamlingen?
        shownWord = listOrder[listOrderIndex]       sätt frågeordet till värdet som räknaren pekar på.
        return shownWord                            returnera frågeordet
    }//Out of bounds, done                     annars
    excerciseDone = true                            markera övningen som klar
    return ""                                       nollställ frågeordet
}
```

4.2 `nextWord` i swift samt pseudokod

Variablerna `listOrder` och `listOrderIndex` lagrar ordningen av frågeorden samt var i ordningen man befinner sig. Detta behövs ifall ett frågeord används flera gånger inom samma övning.

Funktionen `drawNewHand` anropas oftast direkt efter `nextWord`. Den hämtar svarsalternativen associerade med det aktuella frågeordet, slumpar ordningen och returnerar dem.

```
func drawNewHand() -> [String]{
    cardHand.removeAll()                                          ta bort gamla värden från samlingen av svarsalternativ
    if(lists[shownWord] != nil){                                  har frågeordet svarsalternativ?
        var tempList = lists[shownWord]!

        for _ in 0..<lists[shownWord]!.count{                         slumpa ordningen av svarsalternativen
            let rand = Int(arc4random_uniform(UInt32(tempList.count)))    lägg till svarsalternativen till samlingen
            cardHand.append(tempList[rand])
            tempList.remove(at: rand)
        }
    }
    return cardHand                                               returnera samlingen
```

4.3 `drawNewHand` i swift samt pseudokod

Ett svarsalternativ väljs genom att anropa `attemptGuess`. Denna funktion tar svarsalternativet som inparameter och returnerar sant eller falskt beroende på om svaret var korrekt eller inte. Den markerar övningen som avklarad om resultatet blev sant och om detta var sista frågeordet.

```
func attemptGuess(_ guess: String) -> Bool {
        if(lists[shownWord] != nil && keys[shownWord]!.contains(guess)){  finns gissningen med i samlingen av korrekta svarsalternativen?
            if(listOrder.count == listOrderIndex){                            är detta sista frågeordet i övningen?
                excerciseDone = true                                              markera övningen som avklarad
            }
            return true                                                       returnera korrekt
        }                                                                 annars
        return false                                                      returnera inkorrekt
}
```

4.4 `attemptGuess` i swift samt pseudokod

4.3.2 Kortspel

Kortspelet ärver sin grundläggande spellogik från `WordManager` och presenterar denna i form av ett kortspel där spelaren ska para ihop kort för att skapa sammansatta ord. Om kombinationen skapar ett giltigt ord ersätts frågeordet och svarsalternativen med nya ord. Spelet fortsätter tills att alla frågeord i övningen har använts. Kortspelets design går att observera i figur 4.5.

4.5 Illustration av kortspelets användargränssnitt

Svarsalternativen ligger längst ned på skärmen. Minst ett av dessa orden kan matchas med frågeordet som ligger uppe i mitten. Frågeorden och svarsalternativen har varsin korthög, som representeras av korten med karaktärerna på. För att svara drar användaren valfritt svarsalternativ till frågeordet. Om svaret är korrekt spelas ljudet av det sammansatta ordet upp, följt av att frågeordet samt svarsalternativen ersätts. Om svaret är inkorrekt kan användaren inte svara igen på en kort tid. Fördröjningen är där för att uppmuntra användaren att tänka genom varje svar, istället för att successivt testa varje svarsalternativ. Övningen fortsätter tills att alla frågeord på nivån är avklarade.

4.3.3 Bingo

Bingo består av två huvuddelar: rutmönstret och frågeordet som går att observera i figur 4.6. Rutmönstret är av storleken 4x4 då detta passade storleken på skärmen bäst. Frågeordet är ordet till vänster i figuren och är det ord som användaren skall svara på.

4.6 Bild på bingoövningen.

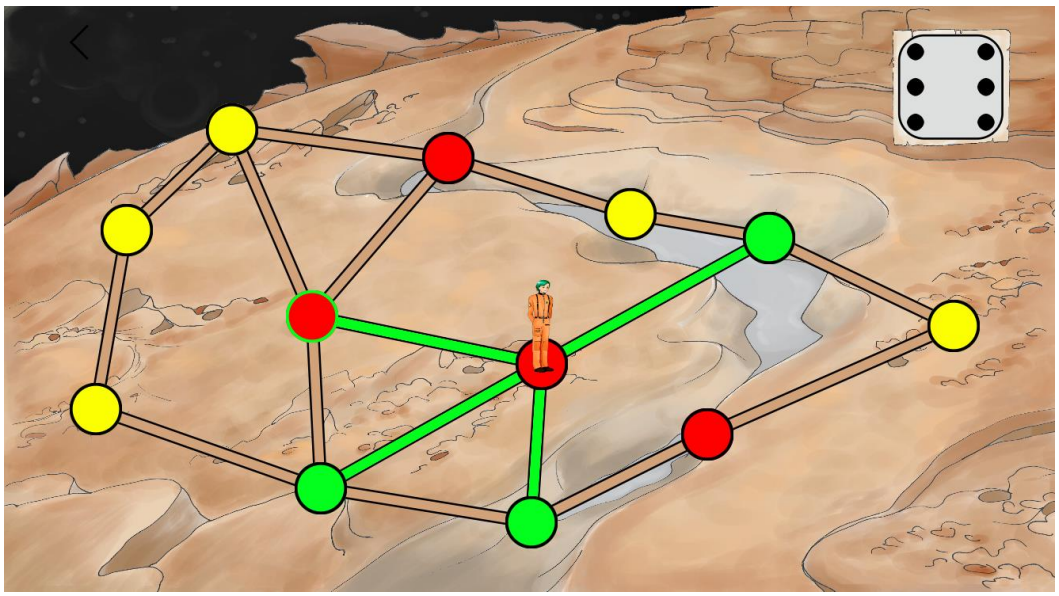
Vid övningens start placeras svarsalternativ på fyra slumpvalda rutor. Ett svar utförs genom att trycka på en ruta med ett svarsalternativ. Om svaret är korrekt sker en animation där frågeordet kombineras med svarsalternativet. Rutan blir ljusare för att markera att den är avklarad. Detta ger en tydlig visuell representation om vad som har skett och det är inga tvivel om att användaren har svarat rätt.

Efter ett korrekt svar kontrollerar spelet om övningen är avklarad. Om ingen rad har fyra avklarade rutor fortsätter övningen med ett nytt frågeord och fyra nya svarsalternativ.

Ett inkorrekt svar gör att svarsalternativet försvinner och dess ruta skakar till. Övningen fortsätter därefter som vanligt, fast med ett svarsalternativ mindre. Om användaren trycker på en ruta utan text förminskas rutan temporärt, för att simulera ett knapptryck. Den finns endast till för att visa att elementet är interaktivt och kan komma att användas senare.

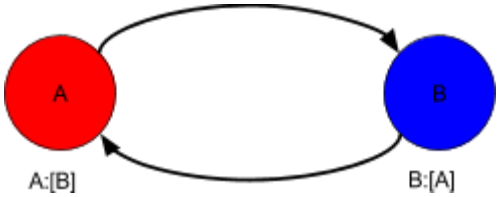
4.3.4 Spelplan

Designen av spelplanen har utformats efter en övning som uppdragsgivaren använder med sina elever. Tanken med övningen är att ge eleven valmöjligheten att skapa sin egen spelplan som eleven sen spelar tillsammans med läraren. Övningen påbörjas med att läraren ber eleven rita ut ett antal spelbrickor på ett papper. Varje spelbricka har ett ord skrivet på sig, under övningens gång kommer eleven besöka alla brickor och skapa ett sammansatt ord på varje bricka elven landar på. Eleven rör sig mellan de ihopkopplade brickorna genom att slå en tärning och gå ett antal steg på spelplanen. Digitalt representeras denna övning som ett Marslandskap med olika knappar som kopplas ihop med vägar samt en tärning som kastas för att avgöra hur långt spelaren får gå, vilket går att se i figur 4.7.



4.7 Ett screenshot från Spelplanen

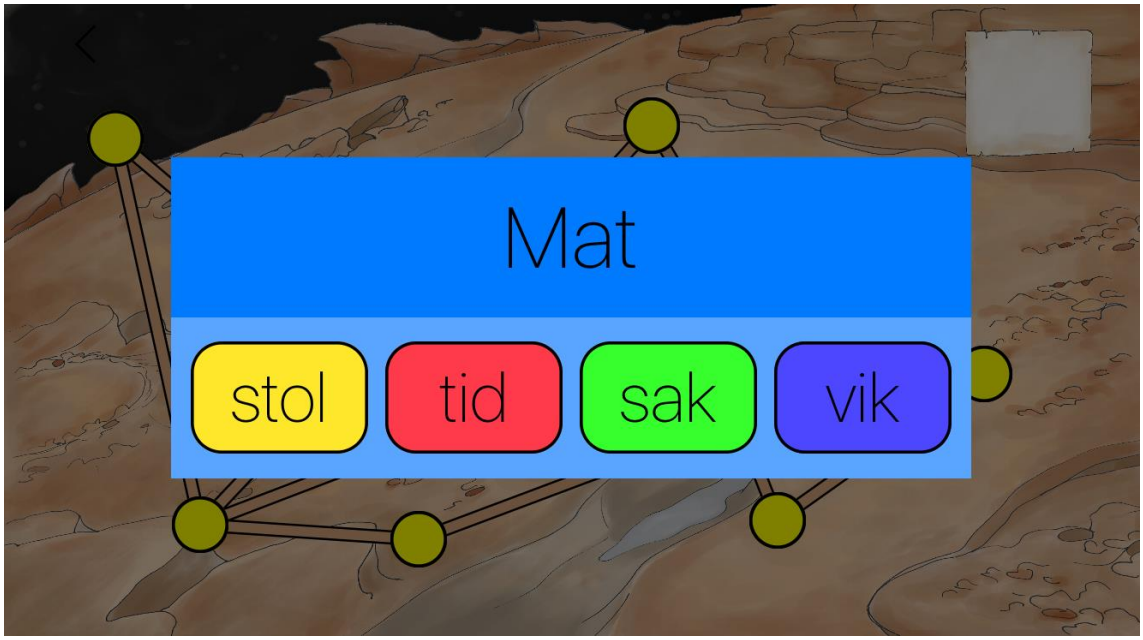
Användaren påbörjar spelet med att placera ut alla spelbrickor. Dessa brickor representeras av runda röda knappar. Den röda färgen på knappen indikerar att knappen inte har en koppling än. När spelbrickorna har placerats ut så skall användaren börja koppla ihop brickorna i valfri ordning vilket ändrar färgen på brickan till gul. En koppling mellan brickor betyder att användarens spelpjäs kan flyttas mellan dessa två brickor. Kopplingen mellan knapparna sker även i den underliggande modellen i form av en dictionary. En dictionary är en nyckel/värde avbildning, i detta fall en [`UIButton`/[`UIButton`]]. Detta betyder att varje nyckel är en utplacerad knapp och dess värde är en avbildning av alla knappar kopplade till nyckeln. Dessa kopplingar är ej riktade vilket betyder att kopplingen måste skapas åt båda hållen. Detta går att se i figur 4.8.



4.8 Illustration på ett oriktat key/value par.

För att starta spelet måste alla brickor ha minst en koppling, men det är möjligt att skapa ytterligare kopplingar för att göra alternativa vägar. När spelaren är redo att börja spela startas spelet på en angiven startbricka.

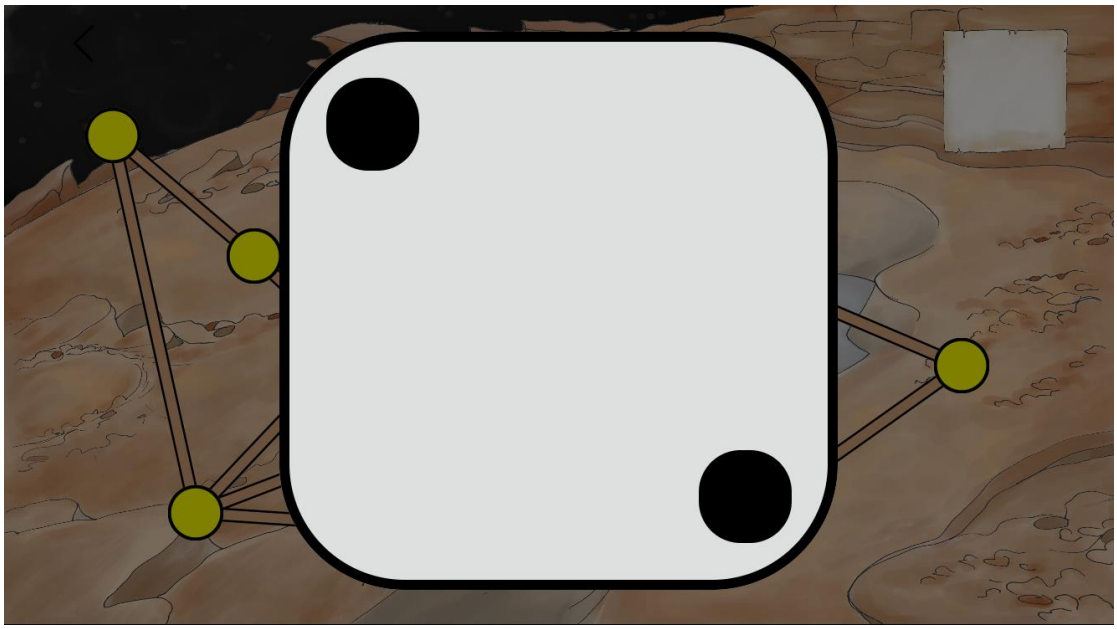
När spelet startar kommer den grundläggande spelloopen att påbörjas. Spelkaraktären placeras på startrutan från början och därmed slås inte tärningen första gången övningen startat utan användaren presenteras direkt med en `WordChoiceView` som går att se i figur 4.9. `WordChoiceView` är designad utifrån samma principer som används i de tidigare spelen, användaren får skapa ett sammansatt ord genom att para ihop ett frågeord med ett av fyra olika alternativ. Detta är huvuddelen av övningen och det är där användaren spenderar majoriteten av sin tid i övningen.



4.9 Ett screenshot från WordChoiceView delen av Spelplanen

Det finns flera olika korrekta kombinationer av ord vilket gör det möjligt för användaren att spela om samma nivå som tidigare och skapa helt nya sammansatta ord. Om användaren skapar ett sammansatt ord som inte finns i facit byter rutan färg från blå till röd och skakar för att indikera att svaret är fel. När användaren har skapat ett korrekt sammansatt ord markeras brickan som användaren står på som klar genom att byta färg från gul till röd och `WordChoice` rutan försvinner. Varje gång ett korrekt ord har skapats spelas en ljudfil som upprepar ordet för användaren. Detta hjälper användare att lära sig uttala de nya sammansatta orden. Efter att användaren har skapat ett korrekt ord är det dags för användaren att flytta sin spelpjäs. På skärmen kommer då en sexsidig tärning att visas upp som användaren kastar för att få reda på hur många steg som spelpjäsen skall förflyttas. Tärningen går att observera i figur 4.10.

När tärningen slås används en rekursiv funktion för att försöka förutse optimala numret för tärningen att landa på. Anledningen till att denna funktion används är för att användaren inte lyckades landa på alla brickor utan att dra ut på tiden med en traditionell tärning. Den rekursiva funktionen använder den tidigare skapade dictionaryn som en tree map och går sedan igenom kopplingarna för att hitta knappar som inte är avklarade och är inom 6 steg från nuvarande knapp.



4.10 Ett screenshot från DiceView delen av Spelplanen

Efter att tärningen har slagits är det upp till användaren att välja hur spelpjäsen skall gå. Målet för användaren är att besöka varje spelbricka och skapa ett sammansatt ord. När alla brickor är markerade som klara är spelet över.

Spelplanen är utvecklad för att vara lätt att förstå och lära in. Detta uppnås genom att använda enkla och klara färger för att förmedla olika händelser och även ett lättförståeligt användargränssnitt vilket visar användaren endast den information som krävs för att klara av övningen. Spelet använder sig även av animationer på både text och bilder för att få användarens uppmärksamhet att fokusera på händelser vilket kräver interaktion.

4.3.5 Belöningssystem

För att hålla användarens intresse över en längre tid skapades ett belöningssystem. Systemet går ut på att användaren får välja en karaktär första gången applikationen startas. Den valda karaktären kommer under spelets gång att få ny utrustning och förbereda sig på en resa till Mars. Denna utrustning är en belöning vilket delas ut efter att användaren har klarat ett visst antal övningar. Applikationen använder inte ett konventionellt belöningssystem med stjärnor eller pokaler eftersom ett antal applikationer på marknaden redan använder detta system. För att få applikationen att särskilja sig från mängden valdes ett system vilket kunde användas inom temat och på så sätt få applikationen att kännas mer som ett spel än ett undervisningsverktyg.

5. Slutsats

Applikationen som har skapats har på ett effektivt sätt lyckats efterlikna de övningar som uppdragsgivaren använder i sin undervisning. Applikationen tillåter eleverna samma valbarhet som finns i de riktiga övningarna och har presenterat dessa med ett innovativt tema för applikationer i denna kategori. Applikationen använder sig av ljudenliga ord och tillhörande ljudfiler för att lära ut materialet. Applikationen har även ett annorlunda belöningssystem som får den att särskilja sig på marknaden jämfört med andra applikationer. Ingen form av mobil uppkoppling krävs för att använda applikationen vilket gör det möjligt att använda den när man reser.

Det finns även förbättringar som kan ske inom applikationen, främst för att förbättra den visuellt men även för att göra den mer lättförståelig. Det saknas en del animationer i kortspelet, till exempel övergången till ett nytt frågeord och när användaren väljer ett inkorrekt svarsalternativ. Animationen på korten som går att dra i känns väldigt stel, vilket kan åtgärdas genom att låta kortet långsamt åka tillbaka istället för att direkt placera tillbaka det på sin ursprungliga plats.

Bingo är den del av projektet som är mest komplett. Övningen har visserligen inte testats av barn i åldern 6-9, men tre vuxna utöver projektgruppen har spelat Bingo. Efter två till tre tryck på skärmen insåg alla hur övningen fungerade. I övrig bemärkelse känns övningen med dess animationer intuitiv och lätt att förstå.

Spelplanen är helt funktionell men det finns saker som går att lägga mer tid på för att göra användarupplevelsen bättre. Att lägga till en swipe gesture för att koppla ihop knappar skulle göra att det känns mer intuitivt att skapa vägar istället för det nuvarande systemet där man klickar. Många delar av övningen skulle kunna uppgraderas utseendemässigt men tiden prioriterades till att få övningen spelbar och buggfri. Expansionsmöjligheter finns i form av nya bakgrunder och ett utökat kopplingssystem. Det finns även intresse för att utöka övningen till att stödja två spelare samtidigt för att kunna spela mot varandra. Detta skulle i så fall ske lokalt då en av målsättningarna med applikationen är att den ej ska vara uppkopplad till något nätverk.

Den bakomliggande modellklassen `WordManager` är i nuvarande stadie duglig. Den är liten och kräver inte att man känner till mycket om den förutom dess funktioner. Det enda fallet som potentiellt kan gå snett är om man anropar `nextWord` och `drawNewHand` i fel ordning. I längden så bör de dictionarys som lagrar informationen om frågeorden bytas ut mot avbildningar, då specifikationen numera hanterar ordnade listor. Denna ändring skulle förenkla `nextWord` avsevärt.

En del av arbetet var att förstå Swift och utvecklingsmiljön Xcode. Swift är relativt likt andra objektorienterade språk, vilket gör att inlärningstiden var relativt snabb. Även upplägget för Xcode var lätt att anpassa sig till. Den större utmaningen låg i att lära sig standardbiblioteket `UIKit` och alla dess funktioner.

Då Xcode stödjer design av användargränssnitt i både `ViewController` och i Storyboards valde projektgruppen att utvärdera de båda alternativen och ta fram fördelar och nackdelar med de båda designalternativen. Storyboards gör det enklare att få en helhetsbild av hur designen kommer att se ut eftersom det går att placera ut de olika `View`-objekten direkt på skärmen. Ett problem med Storyboards är att det inte tillåter samma valbarhet då användaren inte kan transformera lager på samma sätt som koden i en `ViewController`. Att designa programmatiskt kan även vara mindre tidskrävande beroende på designen som skapas. Repetitiva uppgifter, till exempel att placera ut samma objekt på flera olika ställen fungerar bättre i kod än vad det gör via Storyboard.

Ur miljösynpunkt kan denna applikation leda till en mindre förbrukning av papper, då den ersätter ett antal fysiska övningar med en elektronisk tjänst. Den ökade strömförbrukningen i samband med användning av applikationen anses försumbar jämfört med mängden papper som sparas.

6. Referenser

[1] Wikipedia. *Xcode.* Tillgänglig från https://en.wikipedia.org/wiki/Xcode
[2] Developer.Apple. *Model - View - Controller.* Tillgänglig från https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html
[3] Swift.Org. *About Swift.* Tillgänglig från https://swift.org/about/#swiftorg-and-open-source