Inlämningsuppgift 2
”Gissa tal-spel”

Grundläggande programmering

OLOF ALMQVIST
890123–4214

HT 2016

Introduktion

Ett datorspel programmerades i programmeringsspråket C++. Spelet går ut på att spelaren försöker gissa sig fram till ett slumpmässigt framtaget tal mellan 1-50. Spelaren har maximalt 10 gissningsförsök för att erhålla minst ett poäng. Om utmaningen avklaras framgångsrikt erbjuds en vidareutveckling av omgången och man kan välja att spela kvitt eller dubbelt för att antingen dubblera – eller förlora – alla poäng.

I arbetet erhöll programmeraren nya lärdomar om centrala begrepp som fält och strukturer. Två nya tekniker användes även, dels bubbelsortering och dels hantering av inläsning från – samt inskrivning till – textfiler. Dessa två metoder användes för att producera en för upphovsmannen ny funktionalitet i form av en ”High Score”-lista.

Utöver de helt nya lärdomarna har även en vidareutveckling skett av en sedan tidigare känd funktionalitet. I denna uppgift har funktioner utnyttjats avsevärt mycket mer än tidigare och i synnerhet returnerande funktioner d.v.s. de som beskrivs med parametrar, tar in värden, gör en beräkning och returnerar ett svar.

Mjukvaruutvecklingen utfördes så mycket som möjligt genom ”defensiv programmering” vilket beskrivs av Wikipedia (u.å.) som:

> *Defensive programming is a form of defensive design intended to ensure the continuing function of a piece of software under unforeseen circumstances.*

Ansträngningar har alltså gjort för att minimera risken för krascher och buggar för att på det sättet generera en så stabil produkt som möjligt. Frukten av arbetet är en programvara som understruken ser som mer avancerad och svårtolkad än föregående inlämningsuppgift. Koden är mer specificerad och erbjuder mer avancerade funktioner än tidigare.

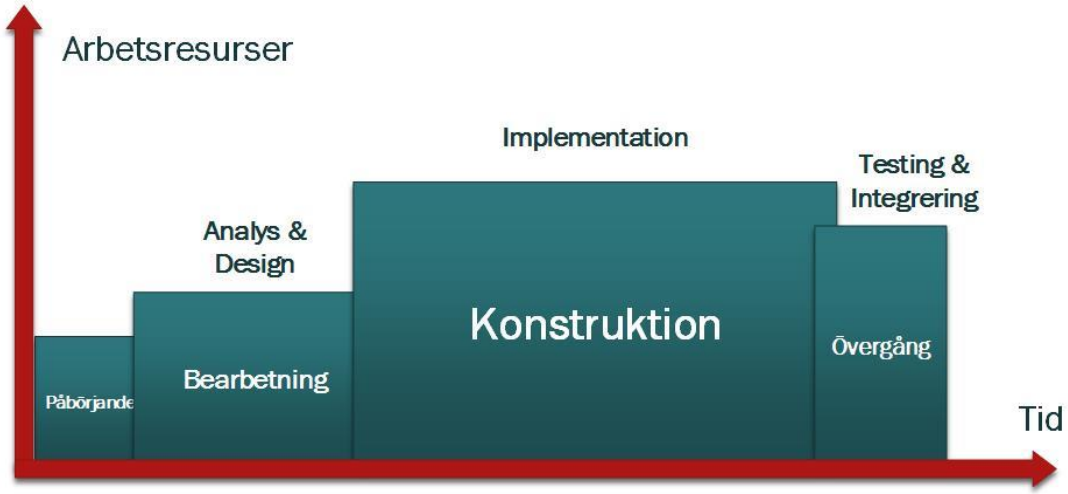
Precis som tidigare har livscykelmodellen använts för att ge struktur i arbetsprocessen. I denna uppgift har den mer generella livscykelmodellen nischats mot en variant som kallas ”Unified Process” (UP) och beskrivits av Scott (2001) som skulle kunna ses som mer allomfattande och beskrivande av hela projektprocessen.

Problembeskrivning

I den initiala fasen av arbetet eller som det beskrivs i UP ”påbörjande-fasen” karaktäriserades av planering av uppgiftens totala omfång. Programmets huvudsakliga problemområden identifierades och beskrevs preliminärt. Svårighetsgraden av de utmaningar som programmeraren ville åta sig att lösa utkristalliserades.

Efter att projektgränserna tagits fram övergick arbetet till fas två i UP-modellen som kallas ”bearbetningsfasen”. Det är detta steg som karaktäriseras genom indelning i faserna ”analys” och ”design” i livscykelmodellen (figur 1).

I denna fas specificerades upphovsmannens egna krav och antaganden som komplement till de obligatoriska som tilldelats i uppgiften.



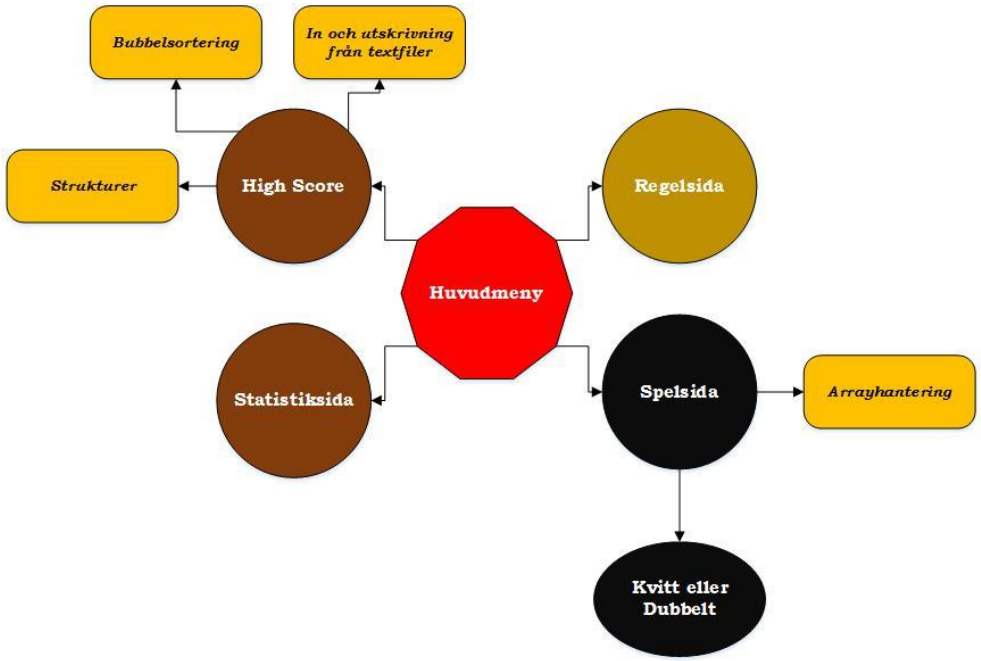
Figur 1. Grafisk illustration av Unified Process-modellen som beskriven av Scott (2001) samt motsvarande steg i livscykelmodellen ovanför boxarna.

I samband med de två första faserna ”påbörjande” och ”bearbetning” gjordes en lista med de konkreta problemområden som programmeraren valde att inkludera i sitt program och planerade att beskriva närmare i ett flödesdiagram. Understruken nu i rollen som mjukvaruarkitekt skissade fram en programbas i form av en huvudmeny som kopplas samman till alla andra delar av programmet. Härifrån går det att navigera vidare till en regelsida som innehåller de antaganden och regler som spelaren behöver känna till för att använda mjukvaran. Vidare finns en statistiksida som visar olika former av nyckeltal som beskriver s.k. ”in game progress”. De två största problemområdena var dels spelsidan som dels innehöll gissa tal mellan 1-50 funktionaliteten samt kvitt eller dubbelt. Dessa områden kräver stor användning av arrayer, vilket var ett nytt koncept för understruken.

Avslutningsvis planerades även en sida som presenterar high score statistik över tid. Här går det att jämföra sina spelresultat mot ett antal förinlagda datorspelare. Informationen som visas här går även att spara mellan speltillfällena och lever därmed kvar i form av en textfil även efter man stänger av programmet.

De olika problemområden som skissades fram i början av arbetet sammanfattas i figur 2.

Det sista mer dolda problemområde som programmeraren önskat ta hänsyn till är det som påverkar programmets prestanda och effektivitet. Så många beräkningar och upprepningar som möjligt skall sammanfattas ner till specifika funktioner, och så få variabler som möjligt skall vara globala för att på så sätt minimera beräkningarnas minnesbelastning på den hårdvara där den exekveras samt ge större kontroll över koden.



Figur 2. De huvudsakliga problemområden som identifierades i förstadiet. Orangea färger beskriver specifika tekniker programmeraren behövde lära sig för att slutföra uppgiften.

Antaganden och krav

I författarens strävan att producera ett tillfredsställande program rörde sig arbetet djupare in i bearbetningsfasen genom livscykelmodellens analyssteg. Arbetet utgick här utifrån de sedan tidigare specificerade obligatoriska krav som kan utläsas i tabell 1.

Ett viktigt problemområde var High Score-sidan samt dess funktionalitet. Här skall spelaren kunna se en lista över de topprankade spelarna genom historien. Listan skall initialt utgöras av på förhand beskrivna spelare och efter hand övertas av de mänskliga spelare som kört programmet.

I det fall att spelet inte tidigare spelats skall ett meddelande visas som påvisar detta faktum för spelaren, och efter att detta gjorts skall ”bas high score”-listan skrivas ut. Detta meddelande kommer alltså enbart upp på skärmen den allra första gången någon går in på high score-sidan. Denna lista skall uppdateras i det fall att spelaren når en genomsnittlig poängnivå som överstiger den lägst rankade datorspelaren. För att hela tiden hålla

listan uppdaterad skall bubbelsortering användas för att sortera high score listan från lägst till högst antal poäng.

På detta vis skall det vara möjligt att se poängräkning mellan de gånger spelet avslutas och jämföra sig själv mot datorn.

Såväl dator- som användare som skrivs in i high score-litsan skall definieras med en struktur. I strukturen skall det finnas två typer av variabler. Dels en integer som beskriver antalet vunna poäng, och dels en string som beskriver spelarens namn.

Regelsidan skall vara en specifik sida i programmet som kan nås via huvudmenyn. På denna sida skall det finnas tillräcklig information för att möjliggöra att spelaren kan förstå hur spelet fungerar på ett sätt som eliminerar risken att missförstånd uppstår eller att spelarens motstånd mot datorn försvåras.

För att ge en förhöjd spelupplevelse skall de olika reglerna skrivas ut genom en fördröjd utskrift på skärmen baserat på antal sekunder, och inte kräva att spelaren trycker på någon knapp. Därtill skall regelsidan avslutas med en minimeny som tillåter snabbare navigering i spelet.

Statistiksidan skall vara en enklare sida som beskriver nyckeltal för spelarens prestanda under spelets gång. Ett av de viktigare talen i detta område skall vara ”Average Points” eller ”Genomsnittliga poäng”. Antal poäng i spelet divideras alltså med antalet ronder. Det är på detta sätt framgång mäts och skrivs in i high score listan eftersom absolut antal poäng skulle innebära att det bara var att spela spelet många gånger för att hamna långt upp.

Spelsidan innehåller själva spelet. Vid valet att gå till spelsidan skall en laddningsskärm visas för att skapa känslan av ett riktigt spel. Ett tal mellan 1-50 skall sedan slumpas fram och spelaren kan gissa max 10 gånger. Understruken är medveten om att kravspecifikationen beskriver att användaren skall kunna gissa hur många gånger som helst men eftersom det inte går att få några poäng efter 10 gissningar förefaller detta snarare vara ett problem och ett slöseri med tid för användaren än en positiv funktionalitet. För att förhöja kvalitén på programmet har systemutvecklaren i detta fall tagit ökad kontroll och gjort ett antagande. Efter varje gissning skall det stå huruvida gissningen var lägre eller högre än det framslumpade talet.

Efter att spelomgången är färdig skall en sammanfattande sida visas. Här skrivs antal spelomgångar ner, antal vunna poäng, huruvida man kan spela kvitt eller dubbelt med mera. Vid val av kvitt eller dubbelt tas 5 tidigare gissade tal fram (eller framslumpade om man gissat färre än 5 gånger i det tidigare spelet). Man får sedan en chans att besvara frågan och om man vinner dubblas poängen, annars förloras samtliga.

Det sista stora kravområdet som tagits fram är det rörande huvudmenyn. Från detta nav i programmet skall det gå att förflytta sig till alla andra stora delar av programmet samt om så önskas avsluta spelsessionen.

Tabell 1. De obligatoriska krav för datorprogrammet som följde med i uppgiftsbeskrivningen.

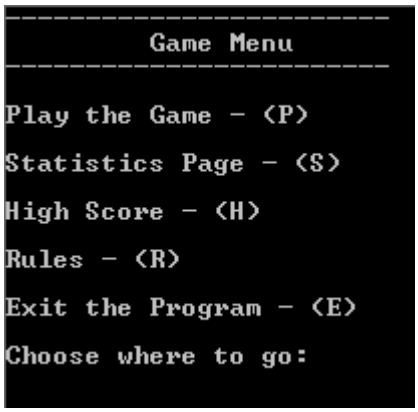
Obligatoriska krav	
Nummer	***Kravbeskrivning***
1	Datorn måste slumpa fram ett tal mellan 1-50 inför varje spelomgång.
2	Tydlig feedback på om användaren gissat rätt eller om det sökta talet är större eller mindre.
3	Användaren kan gissa hur många gången som helst tills det att talet är hittat.
4	Om användaren inte blivit tilldelat något poäng ska det inte vara möjligt att spela ”kvitt eller dubbelt”.
5	Alla tal som användaren har gissat fram måste lagras i en array.
6	Talen som används i spelet ”Kvitt eller dubbelt” får endast bestå av 5st tal som användaren tidigare gissat på. Dessa 5 tal ska lagras i en separat array som sedan visas för användaren.
7	Om användaren förlorar i ”Kvitt eller dubbelt” ska det framgå vilket tal som hade gett vinst.
8	Det ska tydligt framgå hur många poäng som spelaren har vunnit efter varje spelomgång samt hur många poäng som erhållits totalt.
9	Programmet måste vara uppdelat i minst 3st egendefinierade funktioner som består av argument och/eller returnera ett svar. Main är inte en sådan funktion.

Lösningsdesign

Kravspecifikationen nådde en grafisk syntes i form av ett flödesdiagram som ritades vid den tidpunkt då arbetet befann sig mellan bearbetnings- och konstruktionsfas (figur 1). Den färdigställda mjukvaruritningen kan beskådas i bifogad fil ”Flödesdiagram Gissa tal-spel”.

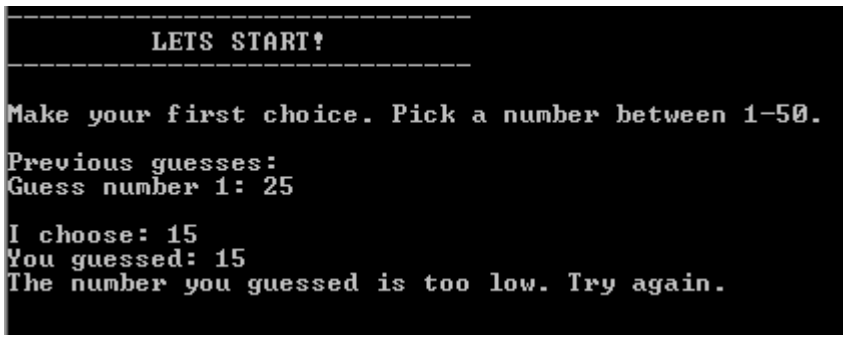
Med detta stöd övergick arbetet helt och hållet till den av UP benämnda konstruktionsfasen eller som livscykeln beskriver den ”implementationsfasen”.

Ett av de första problemområden som hanterades var ***huvudmenyn*** som skrevs i form av en void-funktion (figur 3).



Figur 3. Huvudmenyn som utgör programmets centrala punkt.

Spelsidan skrevs med flera olika fält och funktioner. Ett viktigt krav var att varje gissning skulle returnera en återkoppling angående huruvida gissningen var för låg eller för hög (figur 4).



Figur 4. Den första delen av spelet där man gissar fram tal.

Grunden i spelsidans första del skrevs inom en do-while loop (figur 6). Varje gissning lagras i en variabel ”NumberPick” som sedan laddas i en array som sparar alla tidigare gissningar. I det fall att det gissade talet är högre än 0 och lägre än 51 samt av variabeltypen integer (figur 5) skickas det gissade talet vidare tillsammans med ett sedan tidigare randomiserat vinnartal mellan 1-50 in till en funktion som tar dessa två parametrar och testar om de är likadana eller olika.

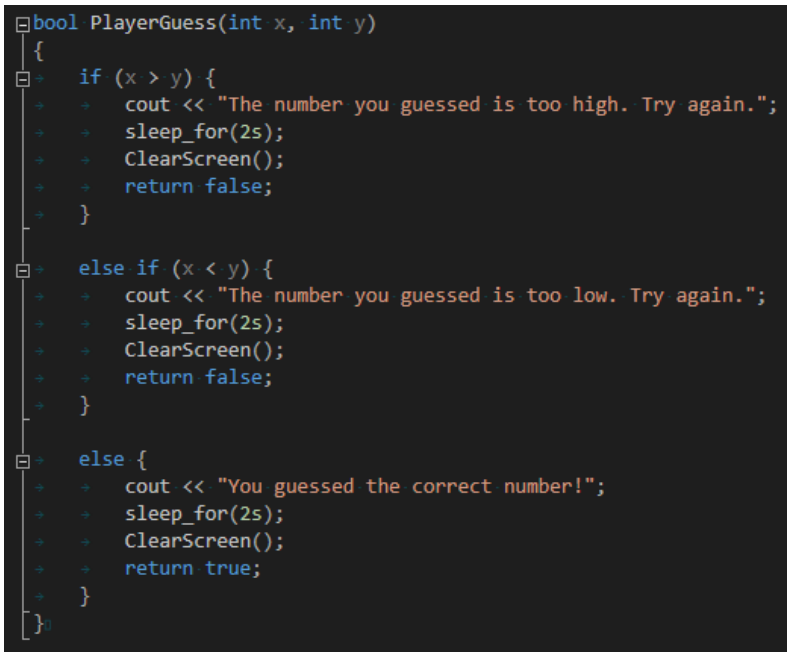
```
cout << endl << "I choose: ";
bool valid = false;

while (!valid) //Defensiv programmering för att rensa bort felaktiga inmatningar
{
    valid = true;
    cin >> NumberPick;

    if (cin.fail() || NumberPick < 1 || NumberPick > 50)
    {
        cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); //förhindrar att ett felmeddelande skrivs ut för varje felaktigt tecken
        cout << "Enter a correct value." << endl;
        valid = false;
    }

}
```

Figur 5. Defensiv kod som förhindrar att felaktiga inmatningar kraschar eller inducerar en bugg i spelet.

```
do
{
    cout << "-----------------------------" << endl;
    cout << "          LETS START!" << endl;
    cout << "-----------------------------" << endl;
    cout << endl << "Make your first choice. Pick a number between 1-50." << endl << endl;

    cout << "Previous guesses: " << endl;

    for (int i = 0; i < NumberGuesses; i++)
    {
        cout << "Guess number " << i + 1 << ": " << GuessedNumbers[i] << endl;
    }

    cout << endl << "I choose: ";
    bool valid = false;

    while (!valid) { ... }

    GuessedNumbers[i] = NumberPick;
    i++;
    cout << "You guessed: " << NumberPick << endl;
    NumberGuesses++;

    TrueOrFalse = PlayerGuess(NumberPick, RandomNumber50);


} while ((TrueOrFalse == false) && (NumberGuesses <= 10));
```

Figur 6. Spelsidans gissa tal-skärm.

Funktionen ”PlayerGuess” tar emot dessa två variabler – gissningen och det framslumpade talet – och testar huruvida gissningen och slumptalet är lika, eller olika varandra (figur 7). Funktionen skickar sedan tillbaka ”true eller false” och baserat på detta returnerade värde fortsätter eller avbryts while-loopen (figur 6, 7)



```
bool PlayerGuess(int x, int y)
{
    if (x > y) {
        cout << "The number you guessed is too high. Try again.";
        sleep_for(2s);
        ClearScreen();
        return false;
    }

    else if (x < y) {
        cout << "The number you guessed is too low. Try again.";
        sleep_for(2s);
        ClearScreen();
        return false;
    }

    else {
        cout << "You guessed the correct number!";
        sleep_for(2s);
        ClearScreen();
        return true;
    }
}
```

Figur 7. Returnerande boolean-funktion som testar spelarens gissning.

I det fall att en korrekt gissning görs och funktionen ”PlayerGuess” därmed validerar ”true” rör sig programmet vidare ner längs koden och ***kallar på en integerfunktion som heter ”GamePointCalc”***. Funktionen tar ett argument som är antalet gissningar. På detta sätt beräknas antalet vunna poäng som returneras tillbaka till spelsidan och lagras i variabeln GamePoints (som kapslas in i void PlayGame).

När poängen mottagits och ett relevant meddelande visats för användaren testas poängen i en ny ***boolean-funktion benämnd ”DoubleorNothing”.*** Denna funktion tar variabeln ”GamePoints” som argument, d.v.s. den mängd som spelaren erhållit under spelet. I funktionen returneras ”true” som spelaren har ett eller fler poäng samt aktivt väljer att spela kvitt eller dubbelt.

Om spelaren således har ett eller fler poäng samt aktivt väljer att spela kvitt eller dubbelt lagras variabeln ”DoN” med ”true”. I figur 8 visas en do-while loop som säkerställer att kvitt eller dubbelt-arrayen som ska slumpa fram tal alltid innehåller minst fem gissningar. Om färre gissningar gjorts slumpas tal mellan 1-50 in i arrayen.

Denna array – GuessedNumbers – som nu innehåller som minst fem olika tal behandlas sedan i en for-loop för att slumpa fram indexpositionerna i fältet. I den andra for-loopen som ses i figur 8 skrivs sedan den initiala arrayen ”GuessedNumbers” över till en andra array ”DoubleorNothingArray” som tar fem unika indexpositioner. Detta görs för att säkerställa att dublettindex inte laddas in i den nya arrayen som skall användas i spelets andra fas – kvitt eller dubbelt.

```
    bool DoN = DoubleorNothing(GamePoints);

if (DoN == true) {
    if (NumberGuesses < 5) //Om spelaren gissar färre än 5 gånger fylls arrayen upp med fler randomiserade tal.
    {
        do
        {
            GuessedNumbers[NumberGuesses] = rand() % 50 + 1;
            NumberGuesses++;

        } while (NumberGuesses < 5);
    }

    for (int i = 0; i < NumberGuesses; i++) //Kasta om innehållet i arrayen
    {
        int index = rand() % NumberGuesses;
        int temp = GuessedNumbers[i];
        GuessedNumbers[i] = GuessedNumbers[index];
        GuessedNumbers[index] = temp;
    }

    for (int i = 0; i < 5; i++) //kopiera över till en andra array
    {
            DoubleorNothingArray[i] = GuessedNumbers[i];
    }

    int WinningNumber = DoubleorNothingArray[rand() % 4];
```

Figur 8. Huruvida spelaren är legitim att spela double or nothing valideras i funktionen "DoubleorNothing".

Det vinnande talet i kvitt eller dubbelt-spelet selekteras genom att lagra in ”DoubleorNothingArray[rand() % 4]” i en variabel. En enkel selektion används sedan för att se om spelarens valda kvitt eller dubbelt-tal är likadant som det vinnande talet.

En annan void funktion som skrevs var ”void StatisticsPage()” (figur 9). Genom att gå in på denna sida kan man få en översiktlig bild över hur spelsessionen har utvecklats. Överst visas spelarens namn och sedan visas ”TotalGamePoints” eller den totala summa poäng som vunnits under sessionen. Antalet rundor som spelats redovisas även.

En viktig del i spelet är att inte bara vinna många poäng utan snarare att få en hög genomsnittlig poängkvot. Detta visas endast om man spelat fler än noll ronder och beskrivs i variabeln ”AveragePoints”. Det är detta nyckeltal som lägger grunden för high score listan som varit ett stort utvecklingsarbete under det här projektet.

```
void StatisticsPage()
{
    ClearScreen();
cout << "-------------------------------------" << endl;
cout << "          Statistics Page" << endl;
cout << "-------------------------------------" << endl;

cout << endl << "On this page, you can track your game performance throughout a session." << endl;

cout << endl << "Game name: " << PlayerName << endl;

cout << endl << "Thus far, you have recieved: " << TotalGamePoints << " number of points." << endl;
cout << "From " << GameRounds << " number of games." << endl << endl;

if (GameRounds > 0)
{
    AveragePoints = TotalGamePoints / GameRounds;
    cout << "That is an average of: " << setprecision(2) << AveragePoints << " points per game round." << endl;

    cout << endl << "Your best round was " << LeastNumberGuesses << " number of guesses" << endl;
    cout << "Which awarded " << LeastNumberGuessesPoints << " number of points." << endl << endl;
}

system("pause");

Menu();
}
```

Figur 9. Statistiksidan som visar nyckeltal för "in game".

High Score-funktionaliteten i spelet kapslas in i funktionen ”void High Score()”. Både de förinskrivna datorspelare och de aktuella mänskliga användarna av spelet beskrivs i strukturen ”struct spelare” som har två egenskaper. Dels stringvärdet name och dels double AverageGamePoints som innehåller den genomsnittliga poängkvoten (figur 10).

```
void HighScore()
{
    ClearScreen();

    if (GameRounds > 0 && TotalGamePoints > 0) { ... }

    struct spelare
    {
        string Name;
        double AverageGamePoints;
    };

    spelare HighScore[3] = {{"Peter Pan", 5}, {"Kapten Krok", 2}, {PlayerName, AveragePoints}};
```

Figur 10. Den kod som skapar high score-listans spelare genom att använda strukturer.

För att hela tiden hålla high score-listan aktuell behövdes en metod för att sortera de genomsnittliga poängkvoterna från högst till lägst. ***Bubbelsortering valdes ut*** som en lämplig metod och programmerades enligt (figur 11).

```
for (int i = 0; i < 2; i++) //Here is the Bubble Sort
{
    for (int j = i + 1; j < 3; j++)
    {
        int TempNum; //Holds AverageGamePoint temporarily
        string TempName; //Holds Name temporarily

        if (HighScore[i].AverageGamePoints < HighScore[j].AverageGamePoints) //If the number i is smaller than j do
        {
            //How the sorting works
            TempNum = HighScore[i].AverageGamePoints;
            TempName = HighScore[i].Name;
            HighScore[i].AverageGamePoints = HighScore[j].AverageGamePoints;
            HighScore[i].Name = HighScore[j].Name;
            HighScore[j].AverageGamePoints = TempNum;
            HighScore[j].Name = TempName;
        }
    }
}
```

Figur 11. Bubbelsortering ser till att den spelare med högst genomsnittlig poäng alltid är högst upp i listan.

Informationen skrivs inte ut till skärmen utan lagras i variabler och skrivs i stället ut till en textfil – HighScore.txt – som sparas i projektmappen. Detta gjordes för att möjliggöra att rekord finns kvar mellan gångerna som spelet stängs av och sätts på igen (figur 12).

```
*** -- HIGH SCORE LIST --- ***
1. Peter Pan with 5 number of points.
2. Kapten Krok with 2 number of points
3. Olof with 1 number of points.
Press any key to continue . . .
```

Figur 12. High Score listan i konsollen. På bilden har användaren "Olof" listats.

I figur 13 visas hur en selektion initialt utförs för att se om en tidigare rekordlista finns sparad, om den inte existerar skrivs ett meddelande ut till spelaren som berättar att det inte spelats några spel tidigare, och sedan skrivs standardlistan in till en nyskapad textfil.

```
ifstream my_file("HighScore.txt"); //Check if HighScore.txt has already been written to the disk.
if (!my_file)
{
    cout << "You have not played any previous games yet." << endl;
    cout << "You can therefor not compare your current progress to your previous." << endl << endl;

    //Here the High Score is written to a text file so it can be saved and uppdated from the players computer.
    ofstream writer("HighScore.txt");
    writer << "*** -- HIGH SCORE LIST --- ***" << endl << endl;
    for (int i = 0; i < 3; i++)
    {
        writer << i + 1 << ". ";
        writer << HighScore[i].Name << " with ";
        writer << HighScore[i].AverageGamePoints << " number of points." << endl << endl;

    }
    writer.close();
}
```

Figur 13. Om ingen tidigare textfil med namnet HighScore.txt finns i projektmappen skrivs standardlistan in i en nyskapad .txt-fil.

För hantering av fler fall kodades två till if-selektioner. Den andra som kan

betraktas i figur 14 skrevs för det fall att en tidigare high score-lista finns samt spelarens snittpoäng är lägre än den sämsta förinlagda datorspelarens poäng. I det fallet skrivs alltså listan ut till skärmen utan att en uppdatering görs.

Den tredje selektionen hanterar det fall att en lista tidigare finns men spelarens poäng är högre än den lägsta datorpoängen. Vid uppfyllelse av det villkoret skrivs den gamla listan först över och läses sedan in på konsolskärmen.

```
else if (my_file && HighScore[3].AverageGamePoints > AveragePoints)
{
    std::string line_;
    ifstream file_("HighScore.txt");
    if (file_.is_open())
    {
        while (getline(file_, line_))
        {
            cout << line_ << '\n';
        }
        file_.close();
    }
    std::cin.get();
}
```

Figur 14. Else if-selektion för om en rekordlista redan finns sparad samt spelarens snittpoäng är lägre än den sämsta datorspelarens snittpoäng.

En sista nämnvärd implementation som gjordes var ”void RulesPage()”. Det är på denna sida programmets regler och för användaren nödvändiga antaganden att känna till skrevs ner. I ett försök att förhöja spelupplevelsen användes funktionen ”sleep_for(xs)” för att ge en känsla av automatisering.

Diskussion

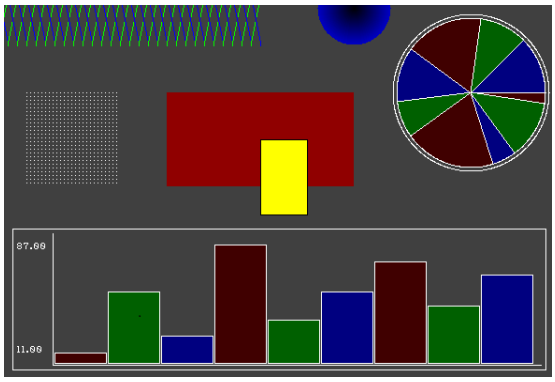
I utvecklingen av programmet utnyttjades flera för programmeraren nya tekniker. Två stora områden som implementerades var bubbelsortering och if- och ofstream för inskrivning och utläsning av textfiler. Vidare användes fält för att skriva spelfunktionaliteten.

Under utvecklingstiden användes på eget bevåg en kombination av ramverket ”Unified Processing” samt livscykelmodellen. I UP-metoden beskrivs implementeringsstadiet som det mest tids- och resurskrävande, någonting som inte riktigt är i linje med underskrivens generella bild av mjukvaruutveckling där en uppfattning finns om att analysfasen är den som ofta tar längst tid (figur 1). Tidsplaneringen har dock passar väl in i beskrivet arbete eftersom kravinsamling och modellering varit relativt små moment i jämförelse med själva implementerings/konstruktionssteget.

En begränsning som finns i programmet är att high score-listor från tidigare spelsessioner (d.v.s. efter att spelet avslutats) endast kan visas en gång. Efter att spelaren producerat ett resultat som skall skivas in i listan skrivs textdokumentet om och därmed försvinner tidigare spel. Ett sätt att lösa det här skulle kunna vara genom att den tidigare listan läses in och sparas i en temporär textfil. Textdokumentet skall alltså läsas in och sparas i temporära variabler för att bevara historik från tidigare gånger då spelet spelats.

En annan förbättring skulle självfallet vara att komplimentera koden till spelet (som kan ses som back end) en ett ”front end” bibliotek. I nuläget interagerar användaren med programmet direkt i konsolen men genom att investera tid och energi i att skriva ett grafiskt användargränssnitt kan en helt annan spelupplevelse skapas.

Ett sätt att utföra detta skulle kunna vara att använda ett enkelt grafiskt c++ bibliotek som ”ezdib” som beskrivits av Umbehant (2012) i figur 14.



Figur 15. Ett exempel på den typ av grafik som skulle kunna kodas i en framtida version med ezdib-biblioteket.

Avslutningsvis önskar författaren understryka att många viktiga lärdomar erhållits under arbetet med spelet och ser nu framför sig en låg karriär av vidareutveckling av de programmeringskunskaper som tillskansats. Stort tack för väldigt bra handledning.

Källkod

```
#include "stdafx.h"
#include <iostream>
#include <thread>
#include <string>
#include <algorithm>
#include <ctime>
#include <fstream> //för high score
#include <iomanip> //För setprecision på doubles

using namespace std::this_thread;
using namespace std;

void Menu();
void ClearScreen();
void RulePage();
void StatisticsPage();
void PlayGame();
void HighScore();

int Randomize50();
int GamePointCalc(int x);

bool DoubleorNothing(int x);
bool PlayerGuess(int x, int y);

string PlayerName;

int GameRounds = 0;
int LeastNumberGuesses = 10;
int LeastNumberGuessesPoints = 0;
double AveragePoints = 0;
double TotalGamePoints = 0;

int main()
{
	ClearScreen();
	cout << setw(55) << "THE AMAZING GUESS THE NUMBERS GAME" <<
endl;
	sleep_for(2s);
	cout << endl << setw(47) << "By Olof Almqvist" << endl;
	sleep_for(4s);

	ClearScreen();
	srand(time(0));
	cout << "-----------------------------------------------" << endl;
	cout << endl << "Welcome to our guess the number game!" << endl;
	cout << "-----------------------------------------------" << endl;
	cout << "Please write your name: "; getline(cin, PlayerName);
```

```
	ClearScreen();
	cout << "Welcome to the game " << PlayerName << endl;
	cout << endl << "Initializing..." << endl;

	for (int i = 0; i < 4; i++)
	{
		sleep_for(0.2s);
		cout << i << "..." << endl;
	}

	Menu();


	return 0;
}

void ClearScreen()
{
	system("cls");
}

//Huvudmenyn
void Menu()
{
	ClearScreen();
	char MenuInput;

	cout << "-------------------------" << endl;
	cout << "        Game Menu" << endl;
	cout << "-------------------------" << endl;
	cout << endl << "Play the Game - (P)" << endl;
	cout << endl << "Statistics Page - (S)" << endl;
	cout << endl << "High Score - (H)" << endl;
	cout << endl << "Rules - (R)" << endl;
	cout << endl << "Exit the Program - (E)" << endl << endl;
	cout << "Choose where to go: "; cin >> MenuInput;

	if ((MenuInput == 'p') || (MenuInput == 'P')) {
		PlayGame();
	}

	else if ((MenuInput == 's') || (MenuInput == 'S')) {
		StatisticsPage();
	}

	else if ((MenuInput == 'h') || (MenuInput == 'H')) {
		HighScore();
	}
```

```
	else if ((MenuInput == 'r') || (MenuInput == 'R')) {
		RulePage();
	}

	else if ((MenuInput == 'e') || (MenuInput == 'E')) {
		ClearScreen();
		cout << endl << "You acquired a total of " << TotalGamePoints << "
number of points." << endl;
		cout << "With an average score of " << AveragePoints << " points
per game. ";
		cout << endl << endl << "Thank you for playing. :)" << endl << endl;
		system("pause");
		exit(0);
	}

	else {

		ClearScreen();
		cout << "You inserted an invalid option. Please try again." << endl;
		sleep_for(1s);
		Menu();
	}
}

//Funktion som innehåller hela spelsidan
void PlayGame()
{
	bool TrueOrFalse = false;
	int NumberPick = 0;
	int NumberGuesses = 0;
	int GamePoints = 0;
	int RandomNumber50 = Randomize50();
	int GuessedNumbers[11], DoubleorNothingArray[4];
	int i = 0;

	GameRounds++;

	for (int i = 0; i < 3; i++)
	{
		ClearScreen();
		cout << "----------------------------" << endl;
		cout << "        Play the Game" << endl;
		cout << "----------------------------" << endl;
		cout << "      Generating numbers" << endl;
		cout << "         [- * -] " << endl;
		sleep_for(0.1s);
		ClearScreen();
		cout << "----------------------------" << endl;
```

```
		cout << "        Play the Game" << endl;
		cout << "----------------------------" << endl;
		cout << "     Generating numbers" << endl;
		cout << "         [- ** -] " << endl;
		sleep_for(0.1s);
		ClearScreen();
		cout << "----------------------------" << endl;
		cout << "        Play the Game" << endl;
		cout << "----------------------------" << endl;
		cout << "     Generating numbers" << endl;
		cout << "         [- *** -] " << endl;
		sleep_for(0.1s);
		ClearScreen();
	}

	do
	{
		cout << "-----------------------------" << endl;
		cout << "        LETS START!" << endl;
		cout << "-----------------------------" << endl;
		cout << endl << "Make your first choice. Pick a number between 1-
50." << endl << endl;

		//Här skrivs tidigare gjorda gissningar ut
		cout << "Previous guesses: " << endl;

		for (int i = 0; i < NumberGuesses; i++)
		{
			cout << "Guess number " << i + 1 << ": " <<
GuessedNumbers[i] << endl;
		}

		cout << endl << "I choose: ";
		bool valid = false;

		while (!valid) //Defensiv programmering för att rensa bort
felaktiga inmatningar
		{
			valid = true;
			cin >> NumberPick;

			if (cin.fail() || NumberPick < 1 || NumberPick > 50)
			{
				cin.clear();

	std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
//förhindrar att ett felmeddelande skrivs ut för varje felaktigt tecken
				cout << "Enter a correct value." << endl;
				valid = false;
```

```
                }

            }

            GuessedNumbers[i] = NumberPick;
            i++;
            cout << "You guessed: " << NumberPick << endl;
            NumberGuesses++;

            //Funktion som jämför det gissade talet mot det framslumpade
vinnartalet
            TrueOrFalse = PlayerGuess(NumberPick, RandomNumber50);

            //loopen fortsätter sålänge antalet gissningar är lägre än 10 samt
rätt tal ej hittats
        } while ((TrueOrFalse == false) && (NumberGuesses <= 10));

        //Funktion för beräkning av antalet vunna poäng (baserat på antalet
gissningar)
        GamePoints = GamePointCalc(NumberGuesses);

        cout << "------------------------" << endl;
        cout << "       AFTERMATH" << endl;
        cout << "------------------------" << endl << endl;
        cout << "You made " << NumberGuesses << " number of guesses to find
the right number." << endl;
        cout << "For that performance, you recieve [ " << GamePoints << " ]
number of points." << endl << endl;
        TotalGamePoints += GamePoints;

        //För statistiksidan. Förmedlar den rond som haft minst antal
gissningsförsök.
        if (NumberGuesses < LeastNumberGuesses)
        {
            LeastNumberGuesses = NumberGuesses;
            LeastNumberGuessesPoints = GamePoints;
        }

        cout << "The numbers you guessed are: ";

        for (int i = 0; i < NumberGuesses; i++)
        {
            cout << GuessedNumbers[i] << ", ";
        }
        cout << endl <<
"_____" << endl <<
endl;

            bool DoN = DoubleorNothing(GamePoints);
```

```
	if (DoN == true) {
		if (NumberGuesses < 5) //Om spelaren gissar färre än 5 gånger
fylls arrayen upp med fler randomiserade tal.
		{
			do
			{
				GuessedNumbers[NumberGuesses] = rand() % 50 +
1;
				NumberGuesses++;

			} while (NumberGuesses < 5);
		}

		for (int i = 0; i < NumberGuesses; i++) //Kasta om innehållet i
arrayen
		{
			int index = rand() % NumberGuesses;
			int temp = GuessedNumbers[i];
			GuessedNumbers[i] = GuessedNumbers[index];
			GuessedNumbers[index] = temp;
		}

		for (int i = 0; i < 5; i++) //kopiera över till en andra array
		{
				DoubleorNothingArray[i] = GuessedNumbers[i];
		}

		//Vinnartalet för kvitt eller dubbelt väljs.
		int WinningNumber = DoubleorNothingArray[rand() % 4];

		int DoubleorNothingPick;

		ClearScreen();
		cout << "------------------------" << endl;
		cout << "    DOUBLE OR NOTHING" << endl;
		cout << "------------------------" << endl << endl;
		cout << "The number has been drawn." << endl;
		cout << "Choose from: ";
		for (i = 0; i < 5; i++)
		{
			cout << DoubleorNothingArray[i];
			if (i < 4)
				cout << ", ";
			else
				cout << ".";
		}

		cout << endl << endl << "Pick your number: "; cin >>
```

```
DoubleorNothingPick;

		//Selektion som validerar om spelaren vann kvitt eller dubbelt
eller ej
		if (DoubleorNothingPick == WinningNumber)
		{
			cout << endl << "You WON! CONGRATULATIONS! :-D" <<
endl << endl;
			TotalGamePoints += GamePoints;
			GamePoints = GamePoints*2;

			cout << endl << "You doubled your points from the guessing
round from " << GamePoints / 2 << " to " << GamePoints << " number of points."
<< endl;
			cout << "Your total score is now: " << TotalGamePoints <<
"." << endl;
			system("pause");
			Menu();
		}

		else
		{
			TotalGamePoints -= GamePoints;
			cout << endl << "Sorry you lost. :-(" << endl << endl;
			cout << endl << "The winning number was: " <<
WinningNumber << ".";
			cout << endl << "You lost all your points, from " <<
GamePoints << " to 0 number of points." << endl << endl;
			cout << "Your total game score is now: " <<
TotalGamePoints << "." << endl << endl;
			system("pause");
			Menu();
		}
	}

	else {
		Menu();
	}

}

//Funktion som tar fram det framslumpade vinnartalet
int Randomize50()
{
	srand(time(NULL));
	int RandomNumber50 = rand() % 50 + 1;

	return RandomNumber50;
}
```

```
//Funktion som testar om spelaren kan & vill spela kvitt eller dubbelt
bool DoubleorNothing(int x)
{
	char choice;

	if (x == 0) {
		return false;
	}

	else {
		cout << "Would you like a chance to double the points you
earned?" << endl << endl;
		cout << "By choosing to play **|--Double or Nothing--|** you can
do just that." << endl << endl;
		cout << "However, if you lose, you will lose all your points." <<
endl << endl;
		cout << "You will be presented with 5 of your previous tries and
pick one." << endl;
		cout << "If you pick the right one, you win, otherwise you lose." <<
endl << endl;
		cout << "Type [Y] to play Double or Nothing." << endl << endl;
		cout << "Type [N] to go back to the menu." << endl << endl;
		cout << "My choice is: "; cin >> choice;

		if ((choice == 'Y') || choice == 'y')
			return true;

		else {
			return false;
		}
	}

}

//Beräkning av antalet vunna poäng samt vinst- eller förlusmeddelanden
int GamePointCalc(int x)
{
	if (x == 1) {
		for (int i = 0; i < 2; i++)
		{
			cout << "-----------------------------------------------" << endl;
			cout << "              GREAT SUCCESS" << endl;
			cout << "-----------------------------------------------" << endl <<
endl;
			cout << "   YOU WON THE MAXIMUM NUMBER OF
POINTS!!!" << endl;
			cout << endl << "       d|^_^|b /" << endl;
			sleep_for(0.5s);
```

```
                        ClearScreen();
                        sleep_for(0.1s);

                        cout << "-----------------------------------------------" << endl;
                        cout << "              GREAT SUCCESS" << endl;
                        cout << "-----------------------------------------------" << endl <<
endl;
                        cout << "   YOU WON THE MAXIMUM NUMBER OF
POINTS!!!" << endl;
                        cout << endl << "       d|^_^|b /" << endl;
                        cout << endl << "****-->MONEY";
                        sleep_for(0.5s);
                        ClearScreen();
                        sleep_for(0.1s);

                        cout << "-----------------------------------------------" << endl;
                        cout << "              GREAT SUCCESS" << endl;
                        cout << "-----------------------------------------------" << endl <<
endl;
                        cout << "   YOU WON THE MAXIMUM NUMBER OF
POINTS!!!" << endl;
                        cout << endl << "       d|^_^|b /" << endl;
                        cout << endl << "********-->MONEY";
                        sleep_for(0.5s);
                        ClearScreen();
                        sleep_for(0.1s);

                        cout << "-----------------------------------------------" << endl;
                        cout << "              GREAT SUCCESS" << endl;
                        cout << "-----------------------------------------------" << endl <<
endl;
                        cout << "   YOU WON THE MAXIMUM NUMBER OF
POINTS!!!" << endl;
                        cout << endl << "       d|^_^|b /" << endl;
                        cout << endl << "**********-->MONEY";
                        sleep_for(0.5s);
                        ClearScreen();
                        sleep_for(0.1s);

                        cout << "-----------------------------------------------" << endl;
                        cout << "              GREAT SUCCESS" << endl;
                        cout << "-----------------------------------------------" << endl <<
endl;
                        cout << "   YOU WON THE MAXIMUM NUMBER OF
POINTS!!!" << endl;
                        cout << endl << "       d|^_^|b /" << endl;
                        cout << endl << "**********************-->MONEY";
                        sleep_for(0.5s);
                        ClearScreen();
```

```
                        sleep_for(0.1s);

                        cout << "-----------------------------------------" << endl;
                        cout << "             GREAT SUCCESS" << endl;
                        cout << "-----------------------------------------" << endl <<
endl;
                        cout << "   YOU WON THE MAXIMUM NUMBER OF
POINTS!!!" << endl;
                        cout << endl << "       d|^_^|b /" << endl;
                        cout << endl << "*************************************--
>MONEY";
                        sleep_for(0.5s);
                        ClearScreen();
                        sleep_for(0.1s);

                        cout << "-----------------------------------------" << endl;
                        cout << "             GREAT SUCCESS" << endl;
                        cout << "-----------------------------------------" << endl <<
endl;
                        cout << "   YOU WON THE MAXIMUM NUMBER OF
POINTS!!!" << endl;
                        cout << endl << "       d|^_^|b /" << endl;
                        cout << endl <<
"*********************************************************-->MONEY";
                        sleep_for(0.5s);
                        ClearScreen();
                        sleep_for(0.1s);
                }
                ClearScreen();
                return 10;
        }

        //Om spelaren har mer än 1 men lika med eller mindre än 3 gissningar
        else if ((x > 1) && (x <= 3)) {
                for (int i = 0; i < 5; i++)
                {
                        cout << "------------------------------" << endl;
                        cout << "        GOOD JOB" << endl;
                        cout << "------------------------------" << endl << endl;
                        cout << "   YOU WON 5 POINTS!!!" << endl;
                        cout << endl << "       d|o-o|b /" << endl;
                        sleep_for(0.2s);
                        ClearScreen();
                        sleep_for(0.2s);
                }
                ClearScreen();
                return 5;
        }
```

```
	//Om spelaren gissar fler än 3 men lika med eller mindre än 10 gånger
	else if ((x > 3) && (x <= 10)) {

	cout << "---------------------------------" << endl;
	cout << "        YOU ARE OK" << endl;
	cout << "---------------------------------" << endl << endl;
	cout << "   YOU WON 1 POINT. KEEP TRYING." << endl;
	cout << endl << "      d|O_O|b /" << endl << endl;
	system("pause");
	ClearScreen();

	return 1;
	}

	else {
		cout << "------------------------" << endl;
		cout << "        FAILURE" << endl;
		cout << "------------------------" << endl << endl;
		cout << "You made too many attempts." << endl;
		cout << "More than 10 attempts rewards 0 points and no more
tries." << endl << endl;
		system("pause");
		Menu();
	}
}

//Sidan som visar statistik under spelets gång
void StatisticsPage()
{
	ClearScreen();
cout << "--------------------------------------" << endl;
cout << "        Statistics Page" << endl;
cout << "--------------------------------------" << endl;

cout << endl << "On this page, you can track your game performance throughout
a session." << endl;

cout << endl << "Game name: " << PlayerName << endl;

cout << endl << "Thus far, you have recieved: " << TotalGamePoints << " number
of points." << endl;
cout << "From " << GameRounds << " number of games." << endl << endl;

//Beräkning görs enbart om minst ett spel har spelats.
if (GameRounds > 0)
{
	AveragePoints = TotalGamePoints / GameRounds;
	cout << "That is an average of: " << setprecision(2) << AveragePoints << "
points per game round." << endl;
```

```
	cout << endl << "Your best round was " << LeastNumberGuesses << "
number of guesses" << endl;
	cout << "Which awarded " << LeastNumberGuessesPoints << " number of
points." << endl << endl;
}

system("pause");

Menu();
}

//Regelsidan
void RulePage()
{
	ClearScreen();
	cout << "------------------------" << endl;
	cout << "        Rules Page" << endl;
	cout << "------------------------" << endl;

	cout << endl << "Welcome to the rule page!" << endl;
	cout << endl << "Here, you will find everything you need to know to play
the game efficiently." << endl;
	sleep_for(2s);

	cout << endl << "*** RULE 1 ***" << endl;
	cout << endl << "You must use this software in a gentlemanly manner." <<
endl;
	sleep_for(2s);

	cout << endl << "*** RULE 2 ***" << endl;
	cout << endl << "You play this game by finding a randomized number
between 1-50." << endl;
	cout << "You have a maximum of 10 attempts. If you exceed this amount
of tries," << endl << "you will recieve 0 points." << endl;
	sleep_for(2s);

	cout << endl << "*** RULE 3 ***" << endl;
	cout << endl << "Finding the correct number with one guess awards 10
points." << endl;
	cout << "Finding the correct number with up to three guesses awards 5
points." << endl;
	cout << "Finding the correct number with more than three but less than
eleven guesses," << endl << "awards 1 point." << endl;
	sleep_for(3s);

	cout << endl << "*** RULE 4 ***" << endl;
	cout << endl << "If you win one or more points while playing the game,"
<< endl << "you are eligible to continue playing Double or Nothing." << endl <<
```

```
endl;
	sleep_for(2s);

	cout << endl << "*** RULE 5 ***" << endl;
	cout << endl << "Choosing to play Double or Nothing means you will get
to" << endl << "choose from five numbers." << endl << endl;
	cout << "One of them is going to be the winning number." << endl;
	cout << "If you pick the correct choice, your points will double. " << endl;
	cout << "If you pick the wrong number, you will loose all points." << endl
<< endl;
	sleep_for(2s);

	cout << endl << "*** RULE 6 ***" << endl;
	cout << endl << "The High Score list measures average points and not
absolute points." << endl << endl;

	//Regelmenyn
	cout << "----------------------------------------" << endl;
	cout << "	How would you like to continue?" << endl;
	cout << "----------------------------------------" << endl;

	cout << endl << "Read the Rules again - (R)" << endl;
	cout << endl << "Play the game - (P)" << endl;
	cout << endl << "Go back to Menu - (M)" << endl;
	char RuleMenu;
	cout << endl << "My choice is: "; cin >> RuleMenu;

	if (RuleMenu == 'R' || RuleMenu == 'r')
	{
		RulePage();
	}

	else if (RuleMenu == 'P' || RuleMenu == 'p')
	{
		PlayGame();
	}

	else
	{
		Menu();
	}
}

//High Score-listan
void HighScore()
{
	ClearScreen();

	if (GameRounds > 0 && TotalGamePoints > 0)
```

```
	{
		AveragePoints = TotalGamePoints / GameRounds;
	}

	//Spelarnas info lagras i form av namn och genomsnittligt spelpoäng
	struct spelare
	{
		string Name;
		double AverageGamePoints;
	};

	//Förinlagda datorspelare Peter Pan och Kapten Krok samt den aktuella spelaren
	spelare HighScore[3] = {{"Peter Pan", 5}, {"Kapten Krok", 2}, {PlayerName, AveragePoints}};

	for (int i = 0; i < 2; i++) //Här är bubbelsorteringen
	{
		for (int j = i + 1; j < 3; j++)
		{
			int TempNum; //Håller AverageGamePoint temporärt
			string TempName; //Håller namnet temporärt

			if (HighScore[i].AverageGamePoints < HighScore[j].AverageGamePoints) //Om numret i är mindre än j gör...
			{
				//Hur sorteringen funkar
				TempNum = HighScore[i].AverageGamePoints;
				TempName = HighScore[i].Name;
				HighScore[i].AverageGamePoints = HighScore[j].AverageGamePoints;
				HighScore[i].Name = HighScore[j].Name;
				HighScore[j].AverageGamePoints = TempNum;
				HighScore[j].Name = TempName;
			}
		}
	}

	ifstream my_file("HighScore.txt"); //Kollar om HighScore.txt redan har sparats på disken.
	if (!my_file)
	{
		cout << "You have not played any previous games yet." << endl;
		cout << "You can therefor not compare your current progress to your previous." << endl << endl;

		//Här skrivs High Score-listan ner i en textfil som sparas på spelarens hårddisk
		ofstream writer("HighScore.txt");
```

```
			writer << "*** -- HIGH SCORE LIST --- ***" << endl << endl;
			for (int i = 0; i < 3; i++)
			{
				writer << i + 1 << ". ";
				writer << HighScore[i].Name << " with ";
				writer << HighScore[i].AverageGamePoints << " number of
points." << endl << endl;

			}
			writer.close();
		}


		//Selektion som läser in high score-listan om textfilen existerar sedan
tidigare & datorspelarna har högre poäng än användaren
		else if (my_file && HighScore[2].AverageGamePoints > AveragePoints)
		{
			std::string line_;
			ifstream file_("HighScore.txt");
			if (file_.is_open())
			{
				while (getline(file_, line_))
				{
					cout << line_ << '\n';
				}
				file_.close();
			}
			std::cin.get();
		}

		//Om spelarens AveragePoints är högre än den lägsta datorspelaren
skrivs en ny high score lista ut och läses följdaktligen in
		else if (HighScore[2].AverageGamePoints < AveragePoints)
		{
			ofstream writer("HighScore.txt");
			writer << "*** -- HIGH SCORE LIST --- ***" << endl << endl;
			for (int i = 0; i < 3; i++)
			{
				writer << i + 1 << ". ";
				writer << HighScore[i].Name << " with ";
				writer << HighScore[i].AverageGamePoints << " number of
points." << endl << endl;

			}
			writer.close();

			std::string line_;
			ifstream file_("HighScore.txt");
			if (file_.is_open())
```

```
		{
			while (getline(file_, line_))
			{
				cout << line_ << '\n';
			}
			file_.close();
		}
		std::cin.get();

	}

	system("pause");

	Menu();
}

//Funktion som testar spelarens gissning i det första gissningsspelet och jämför den mot det framslumpade vinnartalet
bool PlayerGuess(int x, int y)
{
	if (x > y) {
		cout << "The number you guessed is too high. Try again.";
		sleep_for(2s);
		ClearScreen();
		return false;
	}

	else if (x < y) {
		cout << "The number you guessed is too low. Try again.";
		sleep_for(2s);
		ClearScreen();
		return false;
	}

	else {
		cout << "You guessed the correct number!";
		sleep_for(2s);
		ClearScreen();
		return true;
	}
}
```

Referenser

Scott, K. (2001). *Overview of the Unified Process.* InformIT, dec 28. http://www.informit.com/articles/article.aspx?p=24671&seqNum=7 [12-12-2016]

Umbehant, R. (2012). Simple two file graphics library for C/C++. https://www.codeproject.com/articles/363908/simple-two-file-graphics-library-for-c-cplusplus [13-12-2016]

Wikipedia. (u.å.). Defensive Programming. https://en.wikipedia.org/wiki/Defensive_programming [20-12-2016]