Gissa talet

Ett program skriven i C++

Inlämningsuppgift 1, Procedurell Programmering
HT 2011

Emma Jonsson (a11emmjo@student.his.se), Webbutvecklare - Programmering

Sammanfattning

I denna rapport får man följa skapandet av ett spel där användaren skall lista ut ett tal som datorn har slumpat fram, och där användaren i och med detta kan satsa pengar. Spelet skrivs i C++kod och man får följa med ända från kraven som uppdragsgivaren ställer, via självaste programmeringsfasen till slutprodukten.

1 Introduktion

Uppgiften var att skapa ett spel med C++ som sedan skulle spelas i terminalmiljö. Spelet skulle gå ut på att användaren fick sätta in pengar på ett spelkonto och därefter välja ett belopp att satsa. Beroende på det valda beloppet skulle användaren sedan kunna välja ett intervall och därefter skulle ett tal slumpas fram inom det valda intervallet. Användaren skulle därefter gissa på ett tal och om användaren gissade rätt skulle denne vinna dubbla insatsen. Om användaren däremot gissade fel skulle denne få en ny chans tills dess att antalet berättigade gissningar tog slut och då skulle användaren förlora och därmed mista sin insats. Antalet gissningar som användaren skulle ha skulle bero på det valda intervallet.

2 Krav

När spelet skulle skapas fanns vissa krav som man var tvungen att följa. Några har redan framgått i introduktionen men jag väljer ändå att återupprepa dessa och alltså lista alla krav här nedan.

- Spelet ska skapas med hjälp av C++.
- Vid start av programmet skall användaren kunna sätta in pengar på ett ”spelkonto”. Maxbelopp att sätta in är 5000.
- Om användaren inte har några pengar innestående, måste användaren sätta in pengar innan spelomgången startar.
- Innan användaren väljer intervall skall denne välja ett belopp att satsa. Alternativen skall vara 100, 300 eller 500.
- Spelomgången börjar med att användaren matar in ett intervall. Beroende på vad denne har valt att satsa så skall det finnas olika alternativ att välja på. 100 kr: 1-10 eller 1-100. 300 kr: 1-100 eller 1-1000. 500 kr: 1-1000.
- Alternativ som inte är tillgängliga vid en viss insats skall inte heller vara synliga.
- Beroende på valt intervall skall man ha ett visst antal försök på sig att gissa talet. 1-10: 3 försök. 1-100: 5 försök. 1-1000: 7 försök.
- Oavsett om användaren gissar rätt eller fel skall spelomgången avslutas.
- Användaren får spela hur många gånger som helst baserat på att pengar finns på användarens spelkonto, och användaren ska själv välja när denne vill avsluta spelet.
- Lyckas användaren gissa rätt skall vinst och det totalbelopp på spelkontot skrivas ut.
- Gissar användaren fel skall ett tröstmeddelande och totalbelopp på spelkontot skrivas ut.

- Betalningsmomentet kan symboliseras med att pengar dras från användarens konto. Använd gärna ett heltal som minskar alt. ökar beroende på vinster och förluster.

- Talet som skall listas ut kan tas fram med en randomfunktion som returnerar ett slumpmässigt tal.
- Det procedurella paradigmet skall användas för att lösa uppgiften.

- Byggstenarna sekvens, iteration och selektion skall användas.

3 Egna antaganden

Då man försökte att lösa problemet med hjälp de krav man fått så uppstod många luckor. Där fick man alltså göra egna antaganden för att kunna skapa ett komplett fungerande spel. Nedan listar jag de antaganden som jag har ansett varit nödvändiga för att uppgiften skall bli komplett.

- Användaren behöver kunna läsa reglerna om denne ej har spelat detta eller liknande spel tidigare.

- När användaren har läst reglerna antas det att användaren har förstått. Dvs att användaren går inte vidare förrän denne har läst färdigt.

- Man kan max sätta in 5000 åt gången. Men man kan upprepa insättningen hur många gånger man vill för att nå upp till önskat saldo.

- När man lägger sin insats skall man inte kunna satsa mer än vad man har på sitt spelkonto.

- Om insatsen är 500 behöver man ej välja intervall eftersom det enda möjliga intervallet ju är 1-1000. Därav länkas man direkt vidare till nästa moment.

- För att användaren skall ha en något sånär chans att lista ut talet så bör användaren få en upplysning om att det korrekta talet är antingen högre eller lägre än det senast gissade.

- När spelomgången är slut får användaren möjlighet att sätta in mer pengar.

- Om användaren väljer att avsluta så sätts det kvarvarande saldot in på användarens konto inom 3 dagar.

- Det är lätt att råka trycka fel när man exempelvis skall knappa in svaret på en fråga, därav behövs det ett säkerhetsnät som fångar upp ev. felinmatningar.

4 Problemhantering

För att kunna hantera det stora grundproblemet, som var att skapa spelet, så var man tvungen att bryta ner det i mindre delproblem. Nedan listas de problem som dök upp och hur de löstes under arbetets gång.

4.1 Regler

Problemet var att om användaren inte hade spelat spelet tidigare så skulle denne få möjligheten att läsa spelets regler och därefter gå vidare i spelet. Om användaren däremot redan kunde reglerna skulle denne kunna gå direkt vidare i spelet. Här krävdes alltså en funktion som gjorde att användaren själv kunde välja beroende på sina tidigare erfarenheter. Dessutom krävdes en funktion som hindrade användaren från felinmatning.

Lösningen blev att skapa en if-sats (selektion) där användaren fick möjlighet att skriva ”j” för ja eller ”n” för nej i frågan om denne ville läsa spelreglerna. Då risken fanns för att användaren skulle skriva in fel så lades även ett tredje alternativ in i if-satsen och sedan omgavs det hela med en while-loop (iteration). Resultatet blev att om användaren svarade ja så fick denne läsa spelreglerna och sedan gjordes loopen till falsk, vilket innebar att man gick ur loopen och kunde gå vidare till nästa moment i spelet. Om användaren svarade nej så gjordes loopen falsk och användaren gick direkt vidare till nästa moment i spelet. Om användaren däremot tryckte in ett felaktigt tecken så loopades denne om och fick på nytt försöka svara.

4.2 Sätta in pengar

Problemet var att om användaren hade mindre än 100 kr på sitt spelkonto så skulle användaren bli tvungen att sätta in pengar eftersom lägsta belopp att spela för är 100 kr. Om användaren däremot hade mer än 100 kr på sitt spelkonto skulle användaren frågas om denne ville sätta in mer pengar. Här krävdes alltså både en funktion som kontrollerarde användarens saldo och en funktion som gjorde det möjligt för användaren att välja om denne ville sätta in mer. Till sist krävdes en funktion som gjorde det möjligt för användaren att sätta in pengar och som gjorde att pengarna faktiskt hamnade på användarens konto. Dessutom krävdes det ju en funktion som hindrade användaren från att sätta in ett ogiltigt belopp.

Lösningen blev att skapa en if-sats som kontrollerade om användaren hade tillräckligt med pengar för att spela en omgång. Minsta kravet för att kunna spela var ju 100 kr och därför lades detta in som argument. Om användaren hade tillräckligt med pengar så fick användaren veta saldot på spelkontot och sedan fick användaren frågan om denne ville sätta in mer pengar. Svaret kontrollerades precis som tidigare med ytterligare en if-sats omgiven av en while-loop för att undvika ev. felinmatningar. Om användaren däremot hade för lite pengar så hoppades detta moment över och man gick direkt vidare till att sätta in ett valfritt belopp. Det inskrivna beloppet sparades i en variabel och kontrollerades i sin tur av ytterligare en if-sats omgiven av en while-loop. Om man skrev in ett belopp mellan 1 och 5000 som ju var det giltiga intervallet (och därav blev argumentet till if-satsen) så adderades det inskrivna beloppets värde till saldots värde och loopen gjordes falsk och man gick vidare till nästa moment. Om man däremot skrev in ett icke giltig belopp så talades detta om och sedan fick man försöka igen dvs. loopen gick om. Till sist omgavs allt detta av en while-loop som gjorde det möjligt för användaren att sätta in ytterligare pengar på sitt saldo ända tills användaren var nöjd med resultatet och användaren svarade nej på frågan om ytterligare insättning.

4.3 Lägga sin insats

Problemet var att användaren skulle kunna välja en insats att lägga som skulle vara antingen 100, 300 eller 500. Man skulle dessutom inte kunna satsa mer än man hade tillgängligt på sitt spelkonto.

Lösningen blev att användaren ombads att skriva in sin insats som sparades i en variabel sedan kontrollerades insatsen med en if-sats. Om insatsen var giltig dvs. 100, 300 eller 500, (som sattes som argument) så kontrollerades insatsen med ytterligare en if-sats. Om däremot insatsen var ogiltig så fick man försöka igen. Om insatsen var giltig så kontrollerades den med ytterligare en if-sats för att se till så att användaren inte kunde satsa mer pengar än vad som fanns på spelkontot. Runt det hela sattes en while-loop som gjorde att så länge man skrev in antingen en ogiltig insats eller en insats högre än tillgängligt belopp så fick man försöka igen. Var insatsen giltig och möjlig subtraherades insatsens värde från spelkontots saldo och man gick vidare till nästa moment.

4.4 Välja intervall

Problemet var att beroende på vilket belopp användaren hade satsat så skulle man få upp olika förslag på intervall att välja mellan. Hade användaren satsat 100 skulle alternativen vara 1-10 eller 1-100. Hade användaren satsat 300 skulle alternativen vara 1-100 eller 1-1000. Och hade användaren satsat 500 fanns det egentligen ingen anledning för användaren att välja för det enda möjliga intervallet var ju i så fall 1-1000.

Lösningen blev en if-sats som kontrollerade insatsen. Om användare hade satsat 100 kr blev denne ombedd att skriva in vilket intervall den ville gissa i och fick förslagen 1-10 och 1-100. Användarens svar sparades i en variabel och kontrollerades sedan med en if-sats omgiven av en while-loop. Om svaret var 10 så sattes intervallet på 1-10, loopen gjordes falsk och momentet avslutades. Om svaret var 100 så sattes intervallet till 1-100, loopen gjordes falsk och momentet avslutades. Om användaren däremot skrev in ett ogiltigt intervall så fick denne försöka igen.
Samma scenario utspelade sig med en if-sats omgiven av en while-loop, om användaren hade satsat 300. Alternativen att välja på och som sedan kontrollerades var dock inte desamma. Istället fick man välja att skriva in 100 för intervallet 1-100 eller 1000 för intervallet 1-1000.
Om användaren däremot inte hade satsat vare sig 100 eller 300 så måste ju denne ha satsat 500. Det sista alternativen gav då att intervallet automatiskt sattes till 1-1000 och 1000 sparades automatiskt som intervallvariabelns värde.

4.5 Talet slumpas och antal försök fastställs

Problemet var att ett slumpat tal skulle räknas ut inom det intervall som användaren hade valt och sedan skulle även antalet tillåtna gissningar sättas.

Lösningen blev att det valda intervallet kontrollerades med en if-sats. Om värdet på intervallvariabeln var 10 så slumpades ett tal fram mellan 1 och 10 med hjälp av en funktion och sparades i en variabel och sedan fick försöksvariabeln värdet 3. Därefter informerades användaren om antal försök användaren hade på sig för att lista ut talet i det intervall denne valt. Samma tillvägagångssätt men med andra värden utfördes även för de andra alternativen. Var intervallvariabeln 100 slumpades ett tal mellan 1 och 100 fram och försöksvariabeln fick värdet 5. Var intervallvariabeln 1000 slumpades ett tal mellan 1 och 1000 fram och försöksvariabeln fick värdet 7.

4.6 Gissa talet

Problemet var att användaren skulle gissa på ett tal och om det var samma som det tal som hade slumpats fram skulle användaren vinna dubbla insatsen och sedan informeras om vinst och sitt nya saldo. Om talet däremot var fel skulle användaren få en ny chans att gissa ända tills antalet tillåtna gissningar tog slut och då skulle användaren informeras om sin förlust och sitt kvarvarande saldo.

Lösningen blev att användarens gissning sparades i en variabel, som jämfördes med värdet på talet man skulle lista ut, via en if-sats. Om svaret var korrekt så informerades användaren om vinst och insatsen dubblades och adderades till spelkontots saldo. Därefter skrevs det nya saldot på spelkontot ut. Om svaret däremot var fel så subtraherades 1 från försöksvariabeln. Sedan kontrollerades försöksvariabeln med en if-sats. Om försöksvariabelns värde var 0 så informerades användaren om att denne hade förlorat och sedan skrevs det innestående saldot på spelkontot ut. Om däremot försöksvariabelns värde var >0 så fick användaren veta med hjälp av en if-sats att talet antingen var högre eller lägre än det användaren hade gissat tidigare. Alltsammans omgavs sedan av en while-loop som gjorde det möjligt att gissa flera gånger ändå tills att antal försök var slut.

4.7 Spela igen

Problemet var när användaren hade spelat klart spelomgången så skulle användaren få möjligheten att välja om denne ville spela igen. Om användaren av misstag svarade nej skulle denne få ytterligare en kontrollfråga för att kontrollera att användaren verkligen ville avsluta.

Lösningen blev att precis som i avsnitt 4.1 göra en if-sats omgiven av en while-loop som alltså kunde kontrollera svaret och om användaren gav ett felaktigt svar så fick denne ett nytt försök att svara. Om användaren svarade ja gjordes ”spela igen”-loopen falsk. Om användaren däremot svarade nej så fick användaren frågan om denne verkligen var säker på sitt svar och återigen kontrollerades svaret med en if-sats omgiven av en while-loop. Svarade användaren nej då så gjordes svars-loopen falsk och användaren loopades om till den första frågan om användaren ville spela igen, om användaren däremot svarade ja så gjordes alla loopar falska och spelet avslutades.

4.8 Omspel

Problemet var att när spelomgången var slut så skulle användaren alltså kunna spela om igen.

Lösningen blev att lägga en while-loop runt i stort sett hela spelet. För att vara exakt så sträckte den sig från insättning av pengar på spelkontot till frågan om användaren ville sluta spela och alltså avsluta programmet.

5 Buggar

När man programmerar ett spel så är det lätt att missa felaktiga alternativ som användaren kan tänkas utföra och därför svårt att skapa ”skyddsnät” som verkligen skyddar mot alla tänkbara fel och buggar. De buggar som har dykt upp och åtgärdats under utveckligen av detta spel är följande:

- Vad händer om användaren svarar något annat än ja eller nej vid en fråga?

  Svarar användaren med annan bokstav loopas denne om till frågan. Om användaren däremot svarar med ex. en siffra så hamnar man i en evighetsloop. Detta scenario är alltså inte åtgärdat.

- Vad händer om man försöker gå vidare i spelet men ej har tillräckligt med pengar på sitt spelkonto?

  Detta förhindras av en if-sats som kontrollerar så att saldot på spelkontot är >100 innan man går ur insättningsloopen och kan gå vidare till nästa moment.

- Vad händer om man skriver in felaktigt belopp, insats?

  Om användaren skriver in ett ogiltigt belopp så förhindras detta med en if-sats men om användaren ex. skriver in ett decimaltal eller en bokstav så hamnar man i en evighetsloop. Detta scenario är alltså inte åtgärdat.

- Vad händer om man försöker satsa mer än vad som finns på spelkontot?

  Detta förhindras av en if-sats som kontrollerar så att det inmatade värdet är <saldot.

- Vad händer om man gissar på ett tal utanför det valda intervallet?

  Svaret räknas som ett felaktigt svar och drar av ett försök i försöksräknaren trots att svaret är utanför intervallet.

6 Reflektioner

Det spel som har skapats har varit utav mycket simpel karaktär även om koden inte är riktigt lika simpel och man inser plötsligt hur otroligt mycket arbete och kod som måste ligga bakom ett mer seriöst program.

Det har varit en mycket givande och intressant uppgift, speciellt med tanke på att jag inte hade särskilt mycket förkunskaper sen tidigare. Det jag ändå känner att jag har haft nytta av är mina kunskaper inom HTML-kodning och mina mycket nyförvärvade kunskaper inom kodning av väldigt enkla script i Linux- och Windowsmiljö. Dessa har hjälpt mig att se logiken i C++ även om jag inte från början kunde språket.

Det har även varit både en skrämmande och mycket rolig uppgift. Skrämmande på så vis att vi haft relativt lite tid på oss att sätta oss in i och sedan skriva ett fullt fungerande program i C++kod. Men som sagt även roligt då jag personligen verkligen njuter av att hitta felen i min kod och äntligen få allt att fungera. Sedan är det även roligt att se hur programmets funktioner verkligen växer fram.