Inlämningsuppgift 1
Roulett

Grundläggande programmering med C++

Introduktion
Studenter på Högskolan i Skövde fick i uppgift att skapa sitt eget roulettspel genom att använda problemlösning och programmeringsspråket C++.
Uppgiften innehåller bara krav och inga restriktioner för skaparen vid skapandet av programmet. Det skapar problem för nybörjare men tvingar samtidigt studenten att ta egna beslut, vilket leder till att individen utvecklas.

I ett roulettspel ska det finnas en insats, färger, nummer och ett sätt att slumpmässigt ta fram ett resultat (Grundläggande programmering med C++: Inlämningsuppgift 1: Roulette.). I programmet ska användaren spela på ett fiktivt rouletthjul. Spelaren börjar med en fördefinierad summa som kan spelas med. Spelaren kan satsa pengar antigen på ett nummer eller på en färg och vinna eller förlora den satsade summan. En vinnande runda på rouletthjulet ska antigen ge tio gånger insatsen för rätt siffra eller två gånger insatsen på rätt färg. Om det fiktiva hjulet landar rätt på antigen färg eller siffra får spelaren tillbaka sin vinst och insats. Om spelaren får slut på pengar är spelet slut.

Problembeskrivning

Användaren

Användaren av programmet skapar stora problem. Eftersom en person inte kan bli kontrollerad måste spelaren vägledas genom programmet. Om användaren gör något de inte ska, kan de orsaka en bug i programmet och i bästa fall händer inget, men i vissa fall kan de krascha programmet. Det vanligaste problemet användare skapar är olika typer av dataförluster, till exempel råkar användaren ta bort filer eller glömmer att ta med filer. Användaren måste följa instruktionerna som anges när de använder programmet.

Indata

Ett problem med inmatning är om man matar in felaktig information till programmet, så kommer programmet inte att fungera. Detta kan vara frustrerande för användaren och kan leda till att programmet måste startas om från början. Ett annat problem med inmatning är att det kan vara svårt att mata in korrekt information om användaren inte har kunskap om hur programmet fungerar eller den uppgift de försöker utföra. Dessutom kan det vara svårt att mata in information i ett program om användaren är okunnig eller om programmet inte är väl designat.

Spela igen

Programmet måste kunna startas om från börjar för att ett spel ska kunna startas igen. Problemet uppstår eftersom kompilatorn läser kod uppifrån och ner, då det inte är programmerat i kompilatorn att börja om efter koden är över. Det kommer då inte göras automatiskt.

Slumpmässig färg och nummer

Ett fysiskt roulettspel beror på yttre omständigheter, som snurrandet av hjulet, för att bestämma resultatet. I den digitala världen kan en dator generera vad som ser ut som ett slumpmässigt nummer, men faktum är att det inte är ett slumpmässigt tal. Det kallas ett pseudoslumpmässigt nummer. Pseudorandomnummer används i många digitala applikationer, allt från tv-spel till kryptografi. De genereras av algoritmer, som är uppsättningar av instruktioner som producerar en sekvens av nummer som ser slumpmässiga ut men faktiskt är förutsägbara.
Detta innebär att chansen att gissa rätt nummer aldrig är 1 på 36. Medan för färg är det bästa du kan hoppas på en 50-procentschans att få rätt färg.

Insats

Målet är att få spelaren att satsa rätt summa. Detta kan dock vara svårt att uppnå. Insatsens värde som satsas måste stämma överens med hänvisningarna från uppgiften. Spelaren kan sakna tillräckligt med pengar för att satsa minimum av 100 kronor. Användaren kan också försöka satsa mer än vad de har, vilket också resulterar i ett fel. Spelaren kan också försöka satsa mer än det maximala beloppet.

Antaganden och krav

Hur mycket pengar får spelaren tillbaka?

I uppgiften står det att spelaren ska erhållas pengarna tillbaka och en vinstsumma som multipliceras av en faktor. I det här fallet skapas det två vägar av val. Beroende på hur koden struktureras, om pengarna dras när satsningen sker eller om det dras vinst eller förlust.
I den ena vägen menas, spelaren vinner en runda och får tillbaka satsningen samt att de får ytterligare en summa som är lika med summan som satsats multiplicerat med en specificerad faktor. Ett exempel på detta är om spelaren satsar 100 kronor får de tillbaka totalt 200 kronor, 100 kronor som satsat och vinstsumman på 100 kronor.
Den andra vägen. Spelaren satsar en summa och vinner, vinst blir då den satsade summan plus den satsade summan multiplicerat med en specifik faktor. Exempel på detta är spelaren satsar 100 kronor, spelaren får tillbaka 100 kronor (insatsen) och sen vinner 200 kronor från insatsen (i detta fall 100*2).

Programmet ska vara simpelt

Genom att skumma igenom uppgiften ser vi många handlingsverb som pekar på simplicitet. Varje runda måste hjulet redovisas, tydligt framgå om användaren vann eller förlorade, skriva ut den aktuella vinst samt den totala vinstsumman och så vidare.
Ett komplext program kommer aldrig att vara lätt att följa, enkla program kommer alltid att vara lättare att förstå. Detta beror på att komplexa program är designade för att göra många saker, och enkla program är designade för att göra en sak. När ett program är designat för att göra många saker är det svårt att förstå hur det fungerar. Medan ett program designat för att göra en sak gör det lättare att förstå hur det fungerar.

Insatser

Kraven ställda i uppgiften säger att spelaren ska kunna satsa antingen 100, 300 eller 500 kronor. Om spelaren bara har 200 kronor och försöker satsa 300, är detta ett negativt värde. Detta händer för att spelaren inte skulle ha tillräckligt med pengar för att täcka. Spelaren ska inte kunna göra insatsen i första hand. En spelare som har mindre pengar att satsa än det beloppet som försöker satsas, skulle också ge ett negativt värde. Mitt antagande är att spelaren inte ska kunna satsa ett belopp som är lägre än deras saldo, och om de gör det, ska de automatiskt förlora.

Lösningsdesign

Användaren

Det finns ingen allmän lösning på användarfel i program. Vissa tillvägagångssätt för att minska användarfel inkluderar att designa användbara gränssnitt, att tillhandahålla tydliga instruktioner och felmeddelanden.

Indata

Ett sätt att lösa problemet med felaktig inmatning är att skapa ett system av kontroll som säkerställer att informationen som matas in i programmet är korrekt. Detta kan göras genom att användaren får ett felmeddelande om de matar in felaktig information eller genom att ge en bekräftelsemeddelande efter användaren har matat in den korrekta informationen.

Programmet kontrollerar först om användarens inmatning är lika med rätt lösning (i detta fall 1, 2 eller 3) efter att användaren har fått skriva in sitt svar. Om svaret är giltigt, skriver programmet inte ut ett meddelande. Om användarens inmatning inte är lika med rätt lösning, skriver programmet ut ett meddelande som säger " Du har valt ett ogiltigt alternativ", skriver ut alternativen igen och fortsätter loopen. Detta fortsätter tills användaren matar in rätt lösning.

```
while(input !=1 && input !=2 && input !=3) {
        std::cout << "Du valde inte ett giltigt alternativ\n";
        std::cout << "Välj antigen alternativ 1, 2 eller 3!\n";
```

```
    std::cin >> input;
}
```

Spela igen

Lösningen på detta problem är att använda en loop. En loop är en av byggstenarna i det procedurella paradigmet som tillåter kod att köras upprepade gånger. Den typen av loop som valdes för att lösa problemet blev do-while-loopen. En while-loop kommer att köra ett kodblock så länge som ett visst villkor är sant. I det här fallet skulle villkoret vara om spelaren har mer pengar än 100 kronor. Om spelaren har mer än 100 kronor kommer koden att fortsätta att köras om inte spelaren skulle avsluta spelet på egenhand. Om spelet är slut kommer koden inom while-loopen sluta köras.

En do-while-loop är lik en while-loop, förutom att koden i do-while-loopen alltid kommer att köras minst en gång. Detta beror på att villkoret kontrolleras efter att koden i loopen har körts, i stället för före. I det här fallet passar en do-while-loop bättre eftersom spelaren alltid spela minst en gång. Spelaren kommer samtidigt att börja med en summa på 1000 kronor vilket är större än 100 kronor, vilket betyder att kravet redan är uppfyllt.

Slumpmässig färg och nummer

Pseudoslumpmässiga nummer används ofta i stället för sanna slumpmässiga nummer eftersom de är enklare och snabbare att generera. Men de är inte så slumpmässiga egentligen eftersom de genereras av en determinerat algoritm. Det betyder att om vi vet algoritmen, kan nummerserien förutsägas. Även om pseudoslumpmässiga nummer inte är så slumpmässiga, kan de ändå vara användbara i många situationer.
I kod kan vi hjälpa problemet genom att använda oss av tid. Att ha ett klockslag som input gör att den matematiska formeln ändras beroende på klockslag. Vilket i sin tur skapar vad som upplevs vara slumpmässigt, men i det i självaverket blir mera förutsägelsebar om man vet algoritmen. Det som kan sägas är att koden aldrig kommer vara detsamma igen, eftersom funktionen skriver ut sekunder sedan 00:00:00 GMT, Jan 1, 1970.
Det finns några olika sätt att generera färger slumpmässigt i ett program. Ett sätt är att generera slumpmässiga nummer och använda dessa nummer för att skapa en färg. Till exempel kan du generera tre slumpmässiga nummer mellan 0 och 255 och använda dessa nummer som röda, gröna och blå värden för en färg. Det här hade varit alldeles för komplicerat i ett program som det här, det hade varit svårt att implantera som nybörjare. Den lösningen hade skapat en färgton om inte numren hade slumpats till en färg, i det här programmet ska det finnas två färger. Vilket gjorde att lösning inte hade varit relevant. Enligt restriktionerna ska färger relateras till jämna och ojämna nummer vilket kan lösa lätt genom att använda samma grund som koden för den slumpmässiga siffran, koden behövde lite modifikation för önskat resultat.

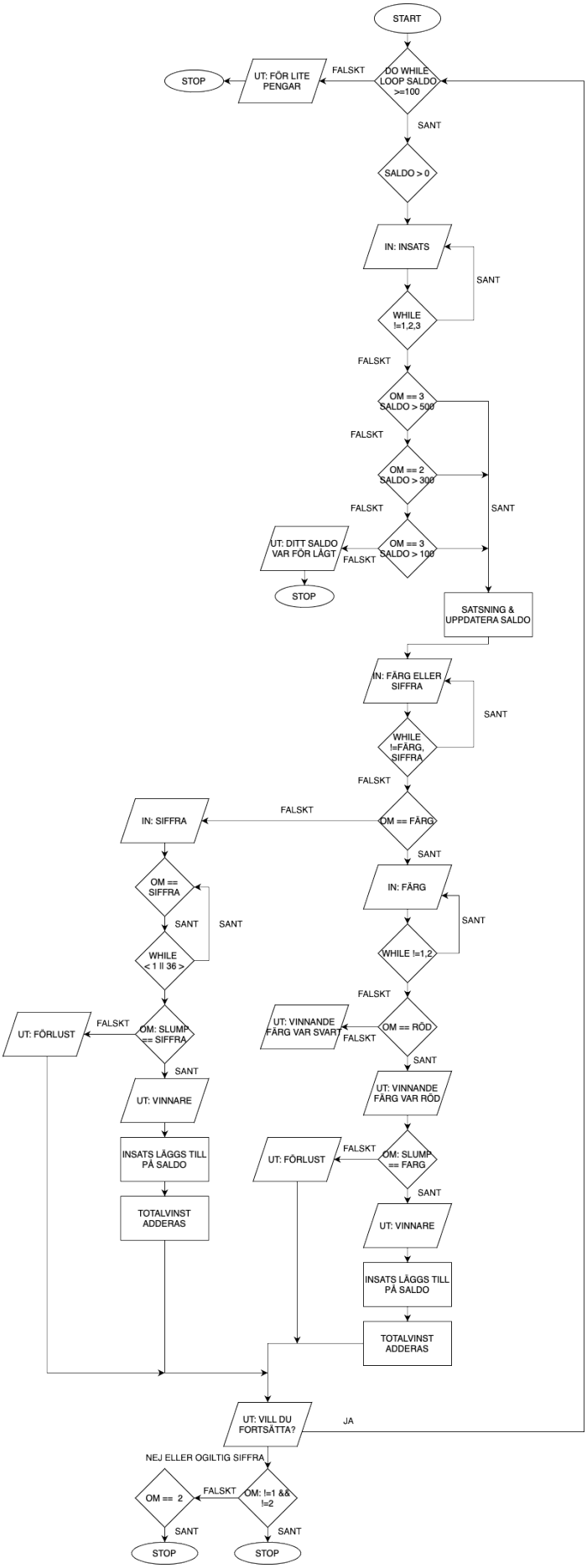
```
srand(time(0));
int slumpmassigsiffra = rand() % 36 +1;
int slumpmassigfarg = (rand() % 2) +1;
```

Insats

En lösning för detta problem kan vara att använda en if-sats. Anledning till användning av en if-sats är att koden inom if-satsen bara körs om villkoret som frågas efter är lika med sant. Villkoren i det här programmet frågar efter vilken input spelaren har gjort. Om spelare har skrivit in ”ett” i terminalen kommer spelare tilldelas 100 kronor att spela med, om spelaren skulle välja ”två” eller ”tre” kommer spelaren tilldelas 300 respektive 500 kronor att spela med. Det betyder att spelaren inte kommer kunna bli tilldelad ett eget valt belopp som antingen överskrider maximumbeloppet eller unserstiger minimumbeloppet. För spelaren ska kunna satsa ett specifikt belopp måste användaren ha mer eller lika med beloppet som satsas.

```
    if (input == 3 && Saldo >=500){
        input = 500;
        Saldo = Saldo - 500;
    } else if (input == 2 && Saldo >= 300){
        input = 300;
```

```
    Saldo = Saldo - 300;
} else if (input == 1 && Saldo >= 100){
    input = 100;
            Saldo = Saldo - 100;
} else break;
```


START
DO WHILE LOOP SALDO >=100
FALSKT
UT: FÖR LITE PENGAR
STOP
SANT
SALDO > 0
IN: INSATS
SANT
WHILE !=1,2,3
FALSKT
OM == 3 SALDO > 500
FALSKT
OM == 2 SALDO > 300
FALSKT
SANT
OM == 3 SALDO > 100
FALSKT
UT: DITT SALDO VAR FÖR LÅGT
STOP
SATSNING & UPPDATERA SALDO
IN: FÄRG ELLER SIFFRA
SANT
WHILE !=FÄRG, SIFFRA
FALSKT
OM == FÄRG
FALSKT
IN: SIFFRA
SANT
IN: FÄRG
OM == SIFFRA
SANT
SANT
WHILE < 1 || 36 >
SANT
WHILE !=1,2
FALSKT
FALSKT
UT: FÖRLUST
OM: SLUMP == SIFFRA
FALSKT
UT: VINNANDE FÄRG VAR SVART
OM == RÖD
FALSKT
SANT
UT: VINNARE
UT: VINNANDE FÄRG VAR RÖD
INSATS LÄGGS TILL PÅ SALDO
FALSKT
UT: FÖRLUST
OM: SLUMP == FÄRG
SANT
TOTALVINST ADDERAS
SANT
UT: VINNARE
INSATS LÄGGS TILL PÅ SALDO
TOTALVINST ADDERAS
UT: VILL DU FORTSÄTTA?
JA
NEJ ELLER OGILTIG SIFFRA
OM == 2
FALSKT
OM: !=1 && !=2
SANT
SANT
STOP
STOP

Diskussion

Spelet är lätt att köra. Programmet har designat från perspektivet av en okunnig spelare, en spelare som aldrig har rört kod, en terminal eller liknande. Varje gång spelaren blir frågad en fråga är programmet tydlig med att man kan välja på alternativ. Om spelaren har skrivit in ett ogiltigt alternativ finns det ett skyddsnät. Det som händer är att spelaren får ett felmeddelande och får skriva in ett giltigt alternativ på nytt. Spelet går igenom spelet steg för steg, det finns aldrig tillfällen där spelare behöver göra mer än en sak åt gången.

Spelet förväntar sig att användaren vet vad roulett är och reglerna. Det som kan sägas är att en tydlig förklaring av hur spelet fungerar saknas. Spelaren får ingen indikation på vad de behöver göra för att vinna spelet. Det finns ingen tydlig målsättning eller objektiv. Utan detta är spelet mer eller mindre en serie av förvirrande val.

Ett utav programmets styrkor ligger i strukturen. Programmet är uppbyggd på ett lätt sätt att läsa. Koden är indragen, vilket förtydligar vad som hänger ihop med vad. Variablerna är tydligt initierade och deklarerade, namnen är relaterade till syftet de har i koden. Koden följer tydligt flödesschemat från topp till tå, det här underlättar för felsökning samt att koden stämmer överens och gör det den ska.

En annan fördel med programmet är att det är lätt att ändra. Om du vill ändra antalet frågor eller vad frågorna är, är det lätt att göra. Du kan också enkelt lägga till nya funktioner, såsom att ta ut pengar eller andra typer av frågor eller funktioner.

En potentiell nackdel med programmet är att den är inte särskilt robust. Om användaren matar in felaktig input, kanske programmet inte fungerar som avsett. Programmet saknar vissa felkontroller. Om användaren har skrivit in en bokstav i stället för siffra kommer programmet att fastna i en oändlig loop tills minnet tar slut. En annan är om spelaren skulle satsa ett belopp större än vad de har i saldo, om spelaren gör det kommer programmet avslutas eftersom den hoppar över alla if-satser.

Källkod

```
//
//  main.cpp
//  roulette
//
//  Created by Hannes Fahlin on 2022-09-09.
//

#include <iostream>
#include <ctime>
#include <cstdlib>

int main() {
    // Programmet startar
    int Saldo = 1000;
    int TotalVinst = 0;
    std::cout << "Ditt saldo är: " << Saldo << " kronor\n";

    // Slumpmässad funktion

    do {
        // Slumpmässig funktion
        srand(time(0));
        int slumpmassigsiffra = rand() % 36 +1;
        int slumpmassigfarg = (rand() % 2) +1;

        if(Saldo > 0) {

            // Välj satsning
            std::cout << "Du kan satsa: [1] 100, [2] 300 eller [3] 500 kronor\n";
```

```
int input = 0;
std::cin >> input;

// Säkerställer att användarens inmatning är giltig
while(input !=1 && input !=2 && input !=3) {
   std::cout << "Du valde inte ett giltigt alternativ\n";
   std::cout << "Välj antigen alternativ 1, 2 eller 3!\n";

   std::cin >> input;
}
// Gör om input till faktiskt valuta och återberättar vad användaren valt
if (input == 3 && Saldo >=500){
   input = 500;
    Saldo = Saldo - 500;
} else if (input == 2 && Saldo >= 300){
   input = 300;
    Saldo = Saldo - 300;
} else if (input == 1 && Saldo >= 100){
   input = 100;
            Saldo = Saldo - 100;
} else break;
std::cout << "Du valde att satsa: " << input << " kronor\n";

int fargellersiffra;
std::cout << "Välj [1] färg eller [2] siffra\n";
std::cin >> fargellersiffra;

// Säkerställer att användarens inmatning är giltig
while (fargellersiffra !=1 && fargellersiffra !=2) {
   std::cout << "Du har valt ett ogiltigt alternativ\n";
   std::cout << "Välj mellan [1] färg eller [2] siffra\n";

   std::cin >> fargellersiffra;
}

// Roulette för färg
if(fargellersiffra == 1){
   int farg;
   std::cout << "Välj en färg mellan [1] Röd och [2] Svart\n";
   std::cin >> farg;

   // Säkerställer att användarens inmatning är giltig
   while (farg !=1 && farg !=2) {
      std::cout << "Du har valt ett ogilgigt alernativ\n";
      std::cout << "Välj mellan [1] Röd eller [2] Svart\n";
      std::cin >> farg;
   }

   if (slumpmassigfarg == farg) {
      std::cout << "Vinnande färg var röd\n";
   } else std::cout << "Vinnande färg var svart\n";

   if (slumpmassigfarg == farg) {
      std::cout << "*******\nVinnare\n*******\n";
      Saldo = (input*2) + Saldo;
      TotalVinst = TotalVinst + input*2;
   } else {
      std::cout << "!!!!!!!\nFörlust\n!!!!!!!\n";
   }
```

```
        }
        else {
            int siffra;
            std::cout << "Välj en siffra mellan [1] och [36]\n";
            std::cin >> siffra;

            // Säkerställer att användarens inmatning är giltig
            while (siffra < 1 || siffra > 36) {
                std::cout << "Du har valt ett ogilgigt alernativ\n";
                std::cout << "Välj en siffra mellan [1] och [36]\n";
                std::cin >> siffra;
            }
            std::cout << "Vinnande nummer: " << slumpmassigsiffra << "\n";
            if (slumpmassigsiffra == siffra) {
                std::cout << "*******\nVinnare\n*******\n";
                Saldo = (input*10) + Saldo;
                TotalVinst = TotalVinst + input*10;
            } else {
                std::cout << "!!!!!!!\nFörlust\n!!!!!!!\n";
            }
        }
        std::cout << "Saldo: "<< Saldo << " kr\n";
        std::cout << "Total vinst " << TotalVinst << " kr\n";
    };

    int fortsatta = 0;
    std::cout << "Vill du fortsätta? [1] Ja [2] Nej\n";
    std::cin >> fortsatta;

    if (fortsatta !=1 && fortsatta !=2) {
        return 0;
    }
    else if (fortsatta == 2) return 0;

} while (Saldo >= 100);

std::cout << "Ditt saldo var förlågt, programmet avslutas.\n";
return 0;
}
```