Inlämningsuppgift 2
”Enarmad bandit”

Grundläggande programmering med C++

Introduktion

Användaren ska spela spelet ”Enarmad bandit”. Spelet enarmad bandit är ett enkelt. En spelare sätter in pengar i maskinen och sedan satsar pengar på varje runda som spelas. När spelet spelas genererar maskinen sedan slumpmässigt tre olika symboler i nio fält. Om spelaren får minst tre av samma symbol i rad vinner de pengar. Summan av pengar som vunnits baseras på symbolerna. Om spelaren får tre olika symboler förlorar de insatsen.
Även om spelet kan verka enkelt finns det mycket komplexitet under huven. De olika symbolerna har olika värden och summan av pengar som vunnits kan variera kraftigt.
Målet är att få så många rader, kolumner och diagonala av samma symbol som möjligt. Om spelaren inte kan få några rader, kolumner eller diagonala av samma symbol, kommer spelaren att förlora spelomgången.

Problembeskrivning

Användaren väljer vad de vill satsa

I programmet ska användaren själv få välja vad de vill satsa. Användaren skulle kunna försöka att skriva in en mängd som är större än beloppet på deras konto. Spelaren skulle kunna göra motsatsens och skriva in ett negativt värde. Om användaren vill använda ett decimaltal medan koden ber om ett heltal kommer talet att rundas ner till närmaste heltal, om indatan är ”3,5” kommer decimaltalet omvandlas till heltalet 3. Det skulle betyda att spelaren förlorar precision om hur de vill spela. I fallet att spelaren vill skriva ut sin insats i bokstäver behöver programmet ha support för den datatypen.

Slumpmässig symbol

Det finns några potentiella problem med att slumpmässigt välja en symbol i programmering. Att slumpa fram en symbol kan vara ett problem eftersom symbol är oftast inte lika tydliga som bokstäver. Ett av problemen med att slumpmässa fram en symbol i programmering är att den inte är inbyggd. Det betyder att om du vill slumpmässa fram en symbol måste du skapa en egen funktion för att göra det. Att skapa en egen funktion kan göra programmet ineffektivt och risken för en bug ökar.

Om bokstaven inte har slumpats ordentligt, kan det innebära att den inte är jämnt fördelad, vilket betyder att vissa bokstäver är mer sannolika att väljas än andra.
Pseudorandomhet är ett matematiskt begrepp som används för att generera till synes slumpmässiga tal. Dessa tal är inte helt slumpmässiga, men de är nära nog slumpmässiga för de flesta ändamål. Problemet med pseudorandomhet är att den inte är helt slumpmässig, och den kan utnyttjas av personer som förstår hur systemet fungerar. I detta fall hade spelaren kunnat veta vilken symbol som vinner.

Sökning av matris för vinst

Det finns några skäl till att sökning i en matris kan vara ett problem. Ett problem är att en matris en tvådimensionell datastruktur, så det kan vara svårt att veta var man ska börja söka efter ett visst element. Ett annat problem kan vara att det en matris ofta lagrad i minnet som en linjär array, vilket innebär att det kan vara svårt att avgöra vilket element som lagras på ett visst index.
Om matrisen är stor, kan sökningen ta lång tid. Om det som söks efter inte finns i matrisen, kan sökningen returnera ett fel. En annat problem kan vara att en array ofta används för att lagra data som inte nödvändigtvis är sorterat (i detta fall är data i arrayen slumpmässigt framkallad), så sökalgoritmen kan behöva söka igenom hela arrayen för att möjligtvis hitta det önskade elementet.
För en vinst, måste spelaren ha minst tre matchande element i rad. Antingen diagonalt, vertikalt eller horisontellt.

Utdata till spelaren

Det finns några potentiella problem som kan uppstå när man skriver ut information till en användare i programmering.
Ett problem kan vara att informationen som skrivs ut är för lång och slutar utanför skärmen. Detta kan

vara frustrerande för användaren, som då måste bläddra igenom mycket text för att hitta den information de behöver. Ett annat problem som kan uppstå är att informationen som skrivs ut är för svår att läsa. Detta kan vara särskilt sant om texten är på ett främmande språk eller använder mycket tekniskt jargon. Slutligen kan ett annat potentiellt problem vara att informationen som skrivs ut är felaktig eller inaktuell, i detta fall om informationen som skrevs ut var från en tidigare runda eller liknande. Detta kan vara vilseledande för användaren och få dem att fatta beslut baserade på felaktig information. Informationen kan vara felformaterad, vilket kan göra den svår att läsa.

Antaganden och krav

Olika symboler

Spelet simulerar en digital enarmad bandit. Spelet behöver någon form av utdata som spelaren kan läsa. I det här fallet måste brädet visas. En möjlighet att visa spelplanen är att använda symboler. Till exempel kan brädet visas som ett rutnät av symboler, där varje symbol representerar ett olika element i spelet.
Detta skulle tillåta spelaren att enkelt se all information de behöver veta om spelet på ett ögonblick. En annan möjlighet att visa spelplanen är att använda text.
Detta skulle tillåta spelaren att se all information de behöver veta om spelet, men det skulle inte vara lika visuellt tilltalande som att använda symboler.

Saldot ska uppdateras

Förutsatt att detta är en enkel version, krävs det att maskinen ska kunna spara spelarens totala saldo och uppdatera detta efter varje snurr på spelautomaten. Dessutom bör maskinen ha någon form av indikator för spelaren om deras aktuella saldo.
Saldot bör uppdateras efter varje snurr på den enarmade banditen av flera skäl. Personen ser uppdateringen av saldot efter varje snurr och samtidigt att maskinen är rättvis och inte ger ut mer pengar än den bör. Detta hjälper till att hålla spelarna från att bli frustrerade och sluta spela spelet.

Om spelet bara skulle visa att spelaren har enbart vunnit eller förlorat skulle spelare lätt kunna bli förvirrad. Att dölja saldot kan ge spelarna en falsk känsla av säkerhet i deras vinster, vilket kan få dem att fortsätta spela tills de har förlorat alla sina pengar, vilket hade ledd till att spelaren hade blivit frustrerad.

Spelare får satsa

Om användaren har tillräckligt med pengar för att satsa, ska spelet tillåta användaren att satsa på omgången. Om användaren inte har tillräckligt med pengar för att satsa, ska spelet inte tillåta användaren att fortsätta spela. Detta är för att försäkra sig om att användaren inte satsar mer pengar än de har.

Lösningsdesign

Användaren väljer vad de vill satsa

Lösningen på detta problem är att begränsa spelarnas kontroll och låta det kunna välja mellan ett fåtal val. Detta kommer att ge användaren möjlighet att välja vad de vill satsa, utan att ha för mycket kontroll över spelet.
Programmet använder sig av ett flertal if-satser. If-satser är ett av de mest grundläggande och vanligt förekommande verktygen i programmering. De gör det möjligt för användaren att köra ett visst kodblock endast om ett visst villkor uppfylls. Detta är mycket användbart för att fatta beslut i programmet, i detta fall kommer det begränsa alternativen spelaren kan satsa.
Spelet skriver ut en text där den ber spelaren skriva in ett heltal för att välja alternativ, det här kommer motverka decimaltalinmatning. Samt att om någon ändå skriver ett decimaltal kommer den bytas till ett möjligtvis giltigt alternativ.

Slumpmässig symbol

För att generera slumpmässiga symboler behöver vi först en lista över alla möjliga symboler som vi kan använda. Detta kan vara allt från bokstäverna i alfabetet till interpunktionsmärken. När vi har

denna lista, kan vi sedan använda en slumpmässig nummergenerator för att välja en symbol från listan slumpmässigt. I det här fallet använder vi bokstäverna X, A och O.
Ett sätt att säkerställa att vår slumpmässiga nummergenerator är mer slumpmässig är att använda funktionen som genererar slumpmässiga nummer till att använda den aktuella tiden som input. Vi kan sedan använda modulus på detta nummer med längden på vår lista med symboler. Detta ger oss ett slumpmässigt index från vår lista med symboler, som vi sedan kan använda för att hämta en symbol slumpmässigt.

```
// Sätter random funktionens input till klockslaget
   long srand(time(0));

   const int rows = 3;
   const int cols = 3;
   char board[rows][cols];

   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         board[i][j] = ' ';
      }
   }

   // Slumpmässa fram där X, A eller O är på brädet
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {

         int r = rand() % 3;
         if (r == 0) {
            board[i][j] = 'X';
         }
         else if (r == 1) {
            board[i][j] = 'O';
         }
         else {
            board[i][j] = 'A';
         }
      }
   }
```

Sökning av matris för vinst
Vi söker efter 3 element i en matris, vi behöver först etablera hur matrisen ser ut och vilka 3 element vi söker efter. Av enkla skäl så säger vi att matrisen är en 2D-array av heltal och de 3 elementen vi söker efter är 1, 2 och 3.
Det finns många olika sätt att söka i en matris efter 3 matchande element i kolumner, diagonala och rader. Ett sätt att göra detta är att använda en nästlad for-loop, en "brute force"-metod för att söka i matrisen. Den yttre for-loopen skulle iterera igenom varje rad i matrisen, och den inre for-loopen skulle iterera igenom varje kolumn i den raden. Om elementet i matrisen på den aktuella raden och kolumnen är lika med elementet i matrisen på nästa rad och kolumn, och också lika med elementet i matrisen på raden efter det, har vi hittat en matchning kommer en tidigare initierade variabel att öka sitt värde med ett. Annars fortsätter vi att leta tills loopen är slut.
Den här loopen behöver göras 3 gånger, en för att söka genom raderna, en för kolumner och en för båda diagonaler.

```
// Räknar hur många raders som hittats
   int found = 0;

   // Söker igenom kolumner
   for (int i = 0; i < rows; i++) {

      for (int j = 0; j < cols; j++) {

         if (board[i][j] == symbol) {
```

```
            if (board[i][j] == board[i][j+1] &&
            board[i][j] == board[i][j+2] && j == 0) {
                found++;
            }
        }
    }
}

// Söker igenom rader
for (int i = 0; i < rows; i++) {

    for (int j = 0; j < cols; j++) {

        if (board[i][j] == symbol) {

            if (board[i][j] == board[i+1][j] &&
            board[i][j] == board[i+2][j] && i == 0) {
                found++;
            }
        }
    }
}

// Söker diagonaler
for (int i = 0; i < rows; i++) {

    // Söker höger ner diagonal
    for (int j = 0; j < cols; j++) {

        if (board[i][j] == symbol) {

            if (board[i][j] == board[i+1][j+1] &&
            board[i][j] == board[i+2][j+2] && j == 0) {
                found++;
            }
        }
    }

    // Söker vänster ner diagonal
    for (int j = 0; j < cols; j++) {

        if (board[i][j] == symbol) {

            if (board[i][j] == board[i+1][j-1] &&
            board[i][j] == board[i-1][j+1] && j == 1 && i == 1) {
                found++;
            }
        }
    }
}
```

Utdata till spelaren

Att man skapar utdata som är läsbar i programmering är viktigt. Först och främst bör utdata vara klar och koncis. Den bör vara fri från onödig information som kan förvirra meddelandet. Därefter bör utdata vara lätt att förstå. Den bör vara skriven på ett sätt som är enkelt och lättillgängligt. Slutligen bör utdata vara visuellt tilltalande. Den bör vara formaterad på ett sätt som är lätt på ögonen och lätt att läsa. Programmet skriver därför ut meddelande med klara och tydliga indikationer på vad som önskas av spelaren.

```
std::cout << "Välj en symbol att satsa pengar på: 'X', 'O' eller 'A' (Skriv i stora bokstäver)\n";

std::cout << "Välj hur mycket pengar du sätter in:\n";
std::cout << "Välj mellan [1] 100, [2] 300 och [3] 500 kronor att sätta in.\n";
```

Diskussion
Det kommandoradapplikation är lätt att förstå från en person som inte är teknisk eftersom applikationen är användarbar. Applikationens grafiska användargränssnitt är lättläst och enkelt att följa. Applikationen har också ett kommandoradinterface som gör det möjligt för användare att skriva kommandon för att interagera med applikationen. Applikationen är utformad för att vara enkel att använda för personer som inte är bekanta med kommandolinjen.

En hashtabell är en datastruktur som lagrar nyckel-värde i par. Det är en snabbare metod för att söka efter ett specifikt värde i en matris än att använda nästlade loopar. Nästlade loopar tar mer tid att söka igenom en matris eftersom de måste kontrollera varje element i matrisen. En hashtabell kan lagra nycklarna i matrisen och de värden som är associerade med dessa nycklar. När en nyckel söks efter returneras det associerade värdet. Detta är snabbare än att använda nästlade loopar eftersom hashtabellen bara behöver söka igenom nycklarna, inte varje element i matris.

Koden är inte är väl organiserad. Den är svår att läsa och förstå vad som händer. Det skulle kunna förbättras genom att ha funktionerna i en annan fil eller genom ett att använda annat paradigm.

Koden är enkel att ändra och det är bra för det innebär att koden kan enkelt modifieras för att passa användarens behov. Detta är särskilt användbart när koden används för ett specifikt ändamål, till exempel ett spel eller ett program som behöver anpassas.

Källkod

```
// Hannes Fahlin

// Hannes Fahlin

#include <iostream>
#include <stdlib.h>
#include <time.h>

int balanceInput();
int stakeInput(int input);
char stakeSymbol();
int randomizedLetter(int input, char symbol);
void success(int input);

int main()
{
   int balance = balanceInput();

   // Loopa medans spelaren fortsätter spela
   while (true) {

      int stake = stakeInput(balance);

      // Tar bort satsningen från totala värdet
      balance = balance - stake;

      char symbol =stakeSymbol();
      int random = randomizedLetter(stake, symbol);
```

```
        success(random);

        // Updaterar värdet på input efter vinst/förlust
        balance = balance + random;
        std::cout << "Du har " << balance << " kronor.\n";

        // Om spelaren får under 100 kronor bryt ut loopen
        if (balance < 100) {
            break;
        }

        std::cout << "Vill du fortsätta spela? Välj [1] för Ja eller [2] för Nej.\n";
        int choice = 0;
        std::cin >> choice;

        if (choice != 1) {
            break;
        }
    }
    std::cout << "Spelet är över";
    return 0;
}

int balanceInput() {
    std::cout << "Välj hur mycket pengar du sätter in:\n";
    std::cout << "Välj mellan [1] 100, [2] 300 och [3] 500 kronor att sätta in.\n";

    // Frågar användaren efter input
    int answer;
    std::cin >> answer;
    int accountBalance = 0;

    // Svar omvandlas till saldo
    if (answer == 1) {
        accountBalance = 100;
    } else if (answer == 2) {
        accountBalance = 300;
    } else if (answer == 3) {
        accountBalance = 500;
    }
    std::cout << "Du har satt in " << accountBalance << " kronor på ditt saldo.\n";

    return accountBalance;

}

int stakeInput(int input) {

    // Bestäm hur mycket som ska satsas
    int choice = 0;
    if (input >= 500) {
        std::cout << "Välj hur slå vad om: " << "[1] 100, [2] 300 eller [3] 500\n";
        std::cin >> choice;

        if (choice == 1) {
            input = 100;
        } else if (choice == 2) {
            input = 300;
        } else {
            input = 500;
```

```
        }

    } else if (input >= 300 && input < 500) {
        std::cout << "Välj hur slå vad om: " << "[1] 100, [2] 200 eller [3] 300\n";
        std::cin >> choice;

        if (choice == 1) {
            input = 100;
        } else if (choice == 2) {
            input = 200;
        } else {
            input = 300;
        }

    } else {
        std::cout << "Välj hur slå vad om: " << "[1] 100 kronor\n";
        std::cin >> choice;

        if (choice == 1) {
            input = 100;
        }
    }

    std::cout << "Du har valt att satsa: " << input << " kronor.\n";

    return input;

}

char stakeSymbol() {

    std::cout << "Välj en symbol att satsa pengar på: 'X', 'O' eller 'A' (Skriv i stora bokstäver)\n";
    char betSymbol;
    std::cin >> betSymbol;
    if (betSymbol == 'X') {
        return betSymbol;
    } else if (betSymbol == 'O') {
        return betSymbol;
    } else if (betSymbol == 'A') {
        return betSymbol;
    } else {
        return -1;
    }
}

int randomizedLetter(int input, char symbol) {

    // Sätter random funktionens input till klockslaget
    long srand(time(0));

    const int rows = 3;
    const int cols = 3;
    char board[rows][cols];

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            board[i][j] = ' ';
        }
    }
```

```
// Slumpmässa fram där X, A eller O är på brädet
for (int i = 0; i < rows; i++) {
   for (int j = 0; j < cols; j++) {

      int r = rand() % 3;
      if (r == 0) {
         board[i][j] = 'X';
      }
      else if (r == 1) {
         board[i][j] = 'O';
      }
      else {
         board[i][j] = 'A';
      }
   }
}

// Skriva ut brädet
for (int i = 0; i < rows; i++) {
   for (int j = 0; j < cols; j++) {
      std::cout << board[i][j];
   }
   std::cout << "\n";
}

// Räknar hur många raders som hittats
int found = 0;

// Söker igenom kolumner
for (int i = 0; i < rows; i++) {

   for (int j = 0; j < cols; j++) {

      if (board[i][j] == symbol) {

         if (board[i][j] == board[i][j+1] &&
         board[i][j] == board[i][j+2] && j == 0) {
            found++;
         }
      }
   }
}

// Söker igenom rader
for (int i = 0; i < rows; i++) {

   for (int j = 0; j < cols; j++) {

      if (board[i][j] == symbol) {

         if (board[i][j] == board[i+1][j] &&
         board[i][j] == board[i+2][j] && i == 0) {
            found++;
         }
      }
   }
}

// Söker diagonaler
for (int i = 0; i < rows; i++) {
```

```
        // Söker höger ner diagonal
        for (int j = 0; j < cols; j++) {

            if (board[i][j] == symbol) {

                if (board[i][j] == board[i+1][j+1] &&
                board[i][j] == board[i+2][j+2] && j == 0) {
                    found++;
                }
            }
        }

        // Söker vänster ner diagonal
        for (int j = 0; j < cols; j++) {

            if (board[i][j] == symbol) {

                if (board[i][j] == board[i+1][j-1] &&
                board[i][j] == board[i-1][j+1] && j == 1 && i == 1) {
                    found++;
                }
            }
        }
    }

    // Utdelning av eventuell vinst
    if (found == 1) {
        input = input * 2;
    }
    else if (found == 2) {
        input = input * 3;
    }
    else if (found == 3) {
        input = input * 4;
    }
    else if (found == 4) {
        input = input * 5;
    }
    else if (found == 5) {
        input = input * 7;
    }
    else if (found == 6) {
        input = input * 10;
    } else {
        input = input - input;
    }

    return input;
}

void success(int input) {
    // Berättar om spelaren vann (i så fall hur mycket) eller förlorade
    if (input == 0) {
        std::cout << "!!!!!Du förlorade!!!!!\n";
    } else {
        std::cout << "*****Du vann " << input << " kronor.*****\n";
    }
}
// Slut på kod /O)__(\)
```