Inlämningsuppgift 2

Enarmad bandit

Grundläggande programmering C++

Roger Johansson

670906-4817

IT120G

Introduktion sid.3

Problembeskrivning sid 3

Antaganden och krav sid 3

Lösningsdesign sid 4

Diskussion sid 8

Källkod sid 8

# **Introduktion**

Uppgiften bestod i att utveckla ett C++-program där en spelare skall kunna spela på en enarmad bandit. Spelaren skall ha ett saldo. Saldot byggs upp av insatser i form av bestämda valörer och utbetalning av eventuella vinster. Slumpen får avgöra vilka symboler av tre olika som kommer fram i spelfältets nio rutor. Vinst utbetalas då minst en rad, diagonal eller kolumn innehåller tre stycken likadana symboler. I uppgiftsbeskrivningen definieras ett antal krav som beskrivs under rubriken *Antaganden och krav*. Under samma rubrik presenteras de antaganden som gjorts i lösningen som inte finns med i kravprofilen.

# **Problembeskrivning**

Följande delproblem skall lösas för att få ett fungerande program enlig kraven:  
Delproblem 1 - Fortsätta eller avbryta  
Programmet måste kunna fortsätta körning tills spelaren väljer att avsluta.   
Programmet måste fråga spelaren om den vill fortsätta spelet.  
Delproblem 2 - Göra och kontrollera insättning   
Programmet måste fråga efter en insättning.

Programmet måste kontrollera att insättningen följer kraven (100, 300 eller 500 kr).  
Delproblem 3 - Göra och kontrollera insats

Programmet måste fråga efter spelarens insats.  
Programmet måste kontrollera spelarens insats.  
Delproblem 4 – Skapa spelfältet  
Programmet måste slumpa fram ett spelfält bestående av symbolerna A, O och X.   
Delproblem 5 – Bestäm antal vinnande rader  
Programmet måste bestämma antalet vinnande rader.  
Delproblem 6 – Presentation av resultat  
Programmet måste presentera det slumpade spelfältet, antalet vinnande rader och resultatet för spelaren (vinst och saldo)

# **Antaganden och krav**

För fullständighetens skull presenteras kortfattat de krav som fanns i uppgiftsbeskrivningen:

* Spelfältet skall presenteras i konsollen
* Spelaren kan fortsätta spela så länge den vill
* Eventuell vinstutbetalning skall presenteras på ett tydligt sätt
* Vinst för rader, kolumner utbetalas enligt i uppgiften angivna odds
* Minst tre egendefinierade funktioner varav en skall ta emot indata och en returnera ett värde
* Spelfältet skall vara uppbyggt av en array

För att lösa uppgiften har följande antaganden gjorts:

* Inga insättningar till spelarens saldo tillåts så länge det finns pengar kvar
* Spelaren får bara satsa hela kronor
* Den enarmade banditen har ingen fast ordning mellan symbolerna utan slumpen styr vilka symboler som utgör spelfältet
* Banken har oändligt mycket pengar.
* Spelarens ålder är utan betydelse.
* Spelaren förstår svenska.
* Felkontroll vid inmatning hålls vid ett minimum.

# **Lösningsdesign**

Utifrån de delproblem som identifierats ovan implementerades en uppgiftslösning vars huvuddrag kommer att beskrivas nedan. För att svenska skall kunna användas krävdes import av biblioteket locale.h och tillägg av kodraden setlocale(LC\_ALL, "sv\_SE"); i main().

Lösning delproblem 1 - Fortsätta eller avbryta  
Frågan om val görs via cout och inmatningen via cin (standardförfarande).   
Programmet körs så länge spelaren vill. Programmet måste aktivt avslutas av spelaren. Detta löstes genom att använda en while(true)-konstruktion runt spelet och spelaren får välja att avsluta efter spelomgången

då det fortfarande finns pengar på saldot:

// fråga om spelaren vill fortsätta spela

playerChoice = continuePlay();

if (!playerChoice) {

cout << "Tack för att du spelade!" << endl;

return 0;

}

eller då pengarna på saldot är slut:

// om pengarna är slut kan spelaren välja att sätta in mer pengar eller avsluta spelomgången

if (playerAccount == 0){

playerChoice = continueDeposit();

if (!playerChoice) {

cout << "Tack för att du spelade!" << endl;

return 0;

}

// sätt in mer pengar

playerAccount = depositPlayerAccount();

}

Lösning delproblem 2 - Göra och kontrollera insättning

På liknande sätt som i lösningen för delproblem 1 får spelaren göra ett val och i det   
här fallet mellan valörerna 100, 300 eller 500 kr. Programmet kontrollerar om insatsen är av dessa valörer. Programmet kör en while-loop tills ett korrekt belopp matats in av spelaren.

// välkomstmeddelande

cout << "Välkommen till den Enarmade Banditen!" << endl;

// fråga efter insättning och spara i spelarens konto

// (flera insatser per spelomgång tillåts inte)

int playerAccount = depositPlayerAccount();

// funktion som kontrollerar att spelaren sätter in ett tillåtet

// belopp (100, 300 eller 500 kr) och returnerar det

int depositPlayerAccount() {

int playerAccountToTest; //variabel som sparar spelarens

// inmatade insatsvärde

while (true) {

cout << "Hur mycket vill du sätta in? (100, 300 eller 500 kr)? ";

cin >> playerAccountToTest;

if (playerAccountToTest == 100 || playerAccountToTest == 300 || playerAccountToTest == 500) {

break;

}

cout << "Du måste sätta in 100, 300 eller 500 kr!" << endl;

}

return playerAccountToTest;

}

Lösning delproblem 3 - Göra och kontrollera insats

Programmet kontrollerar om spelarens insats är mindre eller lika med saldot. Programmet kör en while-loop tills en tillåten insats matats in av spelaren.

// fråga efter insats

int bet = getPlayerBet(playerAccount);

playerAccount -= bet;

// funktion som kontrollerar att spelaren satsar ett tillåtet belopp (hela kronor och tillräckligt med pengar på kontot)

// och returnerar beloppet

int getPlayerBet(int playerAccount) {

int playerBetToTest; //variabel som sparar spelarens inmatade insatsvärde

while (true) {

cout << "Hur mycket vill du satsa? (hela kr)? Du har " << playerAccount << " kr kvar. ";

cin >> playerBetToTest;

if (playerBetToTest <= playerAccount) {

break;

}

cout << "Du kan inte satsa så mycket!" << endl;

}

return playerBetToTest;

}

Lösning delproblem 4 - Skapa spelfältet

Problemet löstes genom att skapa en 2d-array med tre rader och tre kolumner som motsvarar spelfältet. Spelfältet populeras sedan på ett slumpmässigt sätt med symbolerna enligt kraven.

// funktion som bygger upp ett 3 x 3 - fält med slumpmässigt genererade symboler A, X, O

void generatePlayField(char playField[3][3]) {

// loopa över raderna

for (int i = 0; i < 3; i++)

{

// loopa över kolumnerna

for (int j = 0; j < 3; j++)

{

playField[i][j] = charPlayField();

}

}

return;

}

// funktion som returnerar ett tecken för spelfältet. 1 = A, 2 = X

// och 3 = O

char charPlayField() {

int number = generateRandomNumber();

if (number == 1)

{

return 'A';

}

else if (number == 2) {

return 'X';

}

else return 'O';

}

// funktion som genererar ett slumptal där 1 <= slumptalet <= 3

int generateRandomNumber() {

int currentNumber = rand() % 3 + 1;

return currentNumber;

}

Lösning delproblem 5 – Bestäm antal vinnande rader

Problemet löstes genom att skapa en funktion som tar emot ett spelfält (3x3) och räknar rader, kolumner och diagonaler som innehåller samma tecken.

// funktion som returnerar antal vinnande kombinationer för ett

// spelfält

int countWinningCombinations(char playField[3][3]) {

int wins = 0; // variabel som håller antal vinnande

kombinationer

// räkna vinnande rader

for (int i = 0; i < 3; i++)

{

if (playField[i][0] == playField[i][1] && playField[i][1] == playField[i][2])

{

wins++;

}

}

// räkna vinnande kolumner

for (int j = 0; j < 3; j++)

{

if (playField[0][j] == playField[1][j] && playField[1][j] == playField[2][j])

{

wins++;

}

}

// räkna vinnande diagonaler

if (playField[0][0] == playField[1][1] && playField[1][1] == playField[2][2])

{

wins++;

}

if (playField[0][2] == playField[1][1] && playField[1][1] == playField[2][0])

{

wins++;

}

return wins;

}

Lösning delproblem 6 – Presentation av resultat

Programmet presenterar det slumpade spelfältet genom en utskrift till konsollen via funktionen printPlayField().

// funktion som skriver ut spelfältet i konsollen

void printPlayField(char playField[3][3]) {

// loopa över raderna

for (int i = 0; i < 3; i++)

{

// loopa över kolumnerna

for (int j = 0; j < 3; j++)

{

cout << playField[i][j];

}

cout << endl;

}

return;

}

Antalet vinnande rader och resultatet för spelaren (vinst och saldo) presenteras av programmet via funktionen presentResult(). För att matcha antalet vinnande rader med oddsen från kravlistan valdes att använda en dictionary. Dictionaryn kräver att #include <map> läggs till i programmet.

// funktion som presenterar resultatet av spelomgången och betalar ut eventuell vinst.

// returnerar ett uppdaterad värde på spelarens konto

int presentResult(int bet, int wins, int playerAccount) {

// dictionary för att slå upp oddset för ett visst antal rader {antal vinnande rader, odds}

map<int, int> odds = { {1, 2},

{2, 3},

{3, 4},

{4, 5},

{5, 7}

};

if (wins == 0) {

cout << "Tyvärr ingen vinst" << endl;

cout << "Du har nu " << playerAccount << " kr på ditt konto." << endl;

}

else if (wins > 5) {

cout << "Hela fältet!" << endl;

cout << "Du vann " << bet \* 10 << " kr!" << endl;

playerAccount += bet \* 10;

cout << "Du har nu " << playerAccount << " kr på ditt konto." << endl;

}

else {

cout << "Grattis! Antal vinnande rader: " << wins << ". Du vann " << bet \* odds[wins] << " kr!" << endl;

playerAccount += bet \* odds[wins];

cout << "Du har nu " << playerAccount << " kr på ditt konto." << endl;

}

return playerAccount;

}

# **Diskussion**

Styrka: lösningen har hållits enkel, är lätt att följa och uppfyller kraven i problembeskrivningen.

Brist: underdimensionerad felkontroll av inmatade värden.

Brist: int sätter begränsning på inmatade värden och ev. stora vinster.

Brist: om man skulle vilja använda ett spelfält av en annan storlek krävs en del arbete, d.v.s. lösningen kan göras mer generell.

# **Källkod**

Källkoden bifogas i separat dokument.