Egy játék lefejlesztése több tantárgy keretében

Menyhárt László Gábor
(ORCID: 0000-0002-1574-4454)

menyhart@inf.elte.hu
ELTE Eötvös Loránd University, Budapest, Hungary
Faculty of Informatics

Absztrakt. Az ELTE Informatikai Karán a Programtervező informatikus BsC képzés első félévében a hallgatók több gondolkodásmóddal, programozási nyelvvel és azok szintaxisával is megismerkednek. Visszajelzéseik alapján ez – főleg a kezdők számára – gyakran nehézségekbe ütközik, nem látják át az egyes tárgyak közötti összefüggéseket. Ebben a cikkben bemutatok egy lehetséges feladatot és annak különböző módokon előállított megoldásait, melyek segítségével hallgatóinknak bemutathatjuk az egyes tárgyak közötti hasonlóságokat és különbségeket. A bemutatott feladat a jól ismert Blackjack kártyajáték egyszerűsített változata, hogy a hallgatókat jobban tudjuk motiválni és tanórai keretek között feldolgozható legyen.

Kulcsszavak: módszeres programozás, programozási nyelvek, oktatás, szintaxis, összehasonlítás

1. Bevezetés

Az Eötvös Loránd Tudományegyetem Informatikai Karán dolgozom a Média- és Oktatásinformatika Tanszéken, ahol több tantárgyat is tanítok a Programtervező informatikus BSc nappali képzés első évfolyamán. A tantervi hálók [1,2,3] szerint ugyanazt a négy informatikai tárgyat tanulja mind a három specializáción részt vevő összes hallgató. A tantárgyak tematikái [4,5,6,7] és a hallgatók visszajelzései alapján több gondolkodási módot, programozási nyelvet és azok szintaxisát ismerik meg az első félévben. A tapasztalataim és a hallgatókkal folytatott beszélgetések alapján sok kezdő hallgató nehezen birkózik meg ezzel. Sajnos teljesen külön kezelik tárgyakat, és nem hívjuk fel eléggé a figyelmet arra, hogy ez „csak szintaktikai játék", fontos lenne, hogy a programozási nyelvekre eszközként tekintsenek. Elhanyagoljuk az összehasonlításukat, pedig később pont az lesz majd fontos, hogy a megfelelő eszközt válasszák ki az adott probléma megoldásához.

2. Megoldási javaslat

Jó lenne, ha ugyanazt a feladatot lefejlesztenénk több tárgy keretében is, mert így hallgatóink láthatnák, hogy egy problémát többféleképpen is meg lehet oldani. Összehasonlíthatjuk a gondolkodásmódokat és szintaxisokat.

Olyan feladat kell, ami motiválja őket és nem igényel túl nagy fejlesztést, vagyis egy alkalom elég lenne a fejlesztésekhez. Például a 3*45 perces „Programozás" első részén megterveznénk a feladat megoldását, amit a második részén lekódolnának. Ugyanennek a feladatnak a lekódolására már elég lenne a „Számítógépes rendszerek" illetve „Funkcionális programozás" gyakorlatokon a 2*45 perc a kódolásra.

A motiváció miatt valamilyen játékban gondolkoztam és végül az ismert Blackjack kártyajáték egy egyszerűsített verzióját készítettem elő, amit bemutatok most.

Korlátozások

Hogy könnyebb legyen a specifikáció és a fejlesztés beleférjen az időkeretbe az eredeti Blackjack [8] játék csak egy egyszerűsített verzióját dolgozzuk fel. Legyen csak egy pakli kártya, egy osztásban egy

játékos és a gép (Bank) játszanak egymással. Nem engedjük meg a ketté osztásokat. Ha marad idő, vagy vannak érdeklődő hallgatók, akkor házi feladatként is tovább dolgozhatnak rajta.

3. Tervezés

Először „magasabb" szinten tervezzük meg, hogy mi történjen. Azaz részfeladatokra bontjuk a problémát.

```
Keverés
Lapok kiosztása
Játékos laphúzása
Gép (Bank) laphúzása
Pontszámítás
Visszajelzés
```

3.1. Specifikáció

A specifikáció tartalmazza, hogy **mi a kiindulás**, **mi** tartalmazza **az eredmény**t, **mi igaz** a feladat megoldása **előtt és** mi lesz igaz **utána**. A feladat részletesebb specifikációját elkészíthetjük matematikai jelölésekkel, logikai állítások és függvények használatával.

Bemenet:

$const\ N \in \mathbb{N}$, a kártyák száma=52, francia kártya esetén

$deckOfCards \in TCard^N$

$TCard = (color \times numFig \times value)$
$color \in \mathbb{S}, numFig \in \mathbb{S}, value \in \mathbb{N}$, a kártyapakli

Kimenet:

$blackjackResult \in \mathbb{S}$

Előfeltétel:

$\forall i\ (1 \leq i \leq N)\ deckOfCards_i.color \in (heart|diamond|spade|club)$

$deckOfCards_i.numFig$
$\in ("2","3","4","5","6","7","8","9","10","J","Q","K","A")$

$deckOfCards_i.value \in (2,3,4,5,6,7,8,9,10,11)$

Utófeltétel:

Az utófeltételt a 4. fejezetben részletezem a kódok összehasonlításával együtt az érthetőség kedvéért.

3.2. Algoritmus

Az algoritmus tartalmazza azt a folyamatot, hogy **hogyan** jutunk el a kiinduló állapotból a feladat megoldásához. Az algoritmusok készítésétől most eltekintek, mert a különböző környezetek működési módjai, illetve támogatott függvényei és parancsai miatt nem lennének használhatóak mindenhol. Ugyanis a „Programozás" tantárgy keretében előállított algoritmusok a strukturált programozási paradigmát követik ciklusokkal, míg a „Funkcionális programozás" a deklaratívat. Ráadásul magasabb szintű függvényeket, például a rendezést sem kódoltam le.

4. Implementációk összehasonlítása

Az alábbi forráskódok összehasonlításával megismerhetjük a különböző nyelvek és programozási paradigmák eltérő gondolkodásmódjait és szintaxisait. Például láthatjuk, hogy a szekvenciában nincs különbség, az elágazások szintaxisában már láthatunk kis eltérést, viszont a legnagyobb különbség az ismétlésnél van, ahol ugyanazt a műveletet máshogy kell leírni ciklussal illetve rekurzív hívással. A függvények deklarációi és definíciói is mutatnak eltéréseket.

A lefejlesztett kódok a Mellékletben és a [9] linken érhetőek el. Ebben a fejezetben a forráskódok összehasonlítását táblázatos formában mutatom be. Az utófeltétel a kimeneti adatok előállításától hátulról kezdve érthető meg legegyszerűbben, ezért itt is így kezdem a bemutatást.

Mindenhol az utófeltétel megfelelő részével kezdem, amit a `C#` implementáció követ. Ezután a `bash` majd a `Haskell` forráskódok láthatóak.

4.1.

A kimeneti változónak beállítjuk a szöveges válaszokat a végső pontszámok alapján.

$$blackjackResult = \begin{cases} "BLACKJACK\ Win\ -\ 3{:}2" & if\ (playerScoreFinal\ ==\ 21)\ \&\&\ (bankScoreFinal\ <\ 21) \\ "PUSH\ Get\ back\ -\ 1{:}1" & if\ (playerScoreFinal\ ==\ bankScoreFinal)\ \&\&\ (playerScoreFinal\ >\ 0) \\ "BUST\ Win\ -\ 2{:}1" & if\ (playerScoreFinal\ >\ bankScoreFinal) \\ "YOU\ LOST\ -\ 0{:}1" & otherwise \end{cases}$$

```
if ((playerScoreFinal == 21) && (bankScoreFinal < 21)) {
 blackjackResult = "BLACKJACK Win - 3:2";
}
else if ((playerScoreFinal == bankScoreFinal) && (playerScoreFinal >
0)) {
 blackjackResult = "PUSH Get back - 1:1";
} else if (playerScoreFinal > bankScoreFinal) {
 blackjackResult = "BUST Win - 2:1";
} else {
 blackjackResult = "YOU LOST - 0:1";
}
```

```
if [ $playerScoreFinal -eq 21 -a $bankScoreFinal -lt 21 ]; then
 blackjackResult="BLACKJACK Win - 3:2"
elif [ $playerScoreFinal -eq $bankScoreFinal -a $playerScoreFinal -gt
0 ]; then
 blackjackResult="PUSH Get back - 1:1"
elif [ $playerScoreFinal -gt $bankScoreFinal ]; then
 blackjackResult="BUST Win - 2:1"
else
 blackjackResult="YOU LOST - 0:1"
fi
```

```
let blackjackResult = if ((playerScoreFinal == 21) && (bankScoreFinal
< 21))
 then "BLACKJACK Win - 3:2"
 else if ((playerScoreFinal == bankScoreFinal) && (playerScoreFinal >
0))
 then "PUSH Get back - 1:1"
 else if (playerScoreFinal > bankScoreFinal)
 then "BUST Win - 2:1"
 else "YOU LOST - 0:1"
```

4.2.

A végső pontszámokat a kézben tartott kártyák alapján számított összértékekből állítjuk elő figyelembe véve, hogy 21 fölé került-e.

$$finalScore: \mathbb{N} \to \mathbb{N}$$

$$finalScore(x) = \begin{cases} 0 & if\ (x > 21) \\ x & otherwise \end{cases}$$

$$playerScoreFinal = finalScore(playerScore)$$

$$bankScoreFinal = finalScore(bankScore)$$

```
static int finalScore(int score) {
 if (score > 21) {
  return 0;
 } else {
  return score;
 }
}

int playerScoreFinal = finalScore(playerScore);
int bankScoreFinal = finalScore(bankScore);
```

```
function finalScore() {
 if [ $1 -gt 21 ]; then
  echo 0
 else
  echo $1
 fi
}

playerScoreFinal=`finalScore $playerScore`
bankScoreFinal=`finalScore $bankScore`
```

```
finalScore :: Int -> Int
finalScore x
 | x > 21 = 0
 | otherwise = x

let playerScoreFinal = finalScore playerScore
let bankScoreFinal = finalScore bankScore
```

4.3.

A játékosnál és a Banknál lévő kártyákból ugyanúgy számítjuk ki a pontszámokat. Először sorba rendezzük a pontokat, majd összeadjuk őket. Az emelkedő sorrendre azért van szükség, mert így fogjuk tudni a 11 helyett könnyen használni az 1 értéket szükség esetén.

$$playerScore = countScore(sortValue(player_{1..playerN}))$$

$$bankScore = countScore(sortValue(bank_{1..bankN}))$$

```
int playerScore = countScore(playerN, player);
int bankScore = countScore(bankN, bank);
```

```
playerScore=`countScore .player`
bankScore=`countScore .bank`
```

```
let playerScore=countScore player3
let bankScore=countScore bank3
```

4.4.

A számításhoz két függvényt használunk. Az első a rendezés, a második pedig egy összegzés azzal a módosítással, hogy a 11-es értéket 1-nek is tekinthetjük, amennyiben az összegek már 21-nél nagyobb lenne. (Az egyszerűség kedvéért számolunk így, bár egy esetben hibás ez a számítás, amikor két Ász is van, de első beszámításakor éppen 21 pont van. pl.: 10, Ász, Ász)

$$sortValue: TCard^* \to \mathbb{N}^*$$

$$sortValue(X) = values, where$$

$$Length(X) = Length(values) \wedge$$

$$\forall i\big(1 \le i \le Length(X)\big)\exists j\big(1 \le j \le Length(X)\big): X_i.value = values_j \wedge$$

$$\forall i\big(1 \le i \le Length(X)\big)\exists j\big(1 \le j \le Length(X)\big): values_i = X_j.value \wedge$$

$$\forall i(1 \le i \le Length(X) - 1) values_i \le values_{i+1}$$

$$countScore: \mathbb{N}^* \to \mathbb{N}$$

$$countScore(X_0) = 0$$

$$countScore(X_{1..N}) = countScore(X_{1..N-1}) + \begin{cases} 1 & if\ X_N = 11 \wedge countScore(X_{1..N-1}) + X_N > 21 \\ X_N & otherwise \end{cases}$$

```
static int countScore(int N, string[] arr) {
 int[] values = new int[N];
 for (int i = 0; i < N; i++) {
  values[i] = Int32.Parse(arr[i].Split(";")[2]);
 }
 Array.Sort(values);
 int sum = 0;
 for (int i = 0; i < N; i++) {
  int x = values[i];
  if ((x == 11) && (sum + x > 21)) {
   sum += 1;
  } else {
   sum += x;
  }
 }
 return sum;
}
```

```
function countScore() {
 sum=0
 for x in `cat $1 | cut -f 3 -d";" | sort -n`; do
  if [ $x -eq 11 -a `expr $sum + $x` -gt 21 ]; then
   sum=`expr $sum + 1`
  else
   sum=`expr $sum + $x`
  fi
 done
 echo $sum
}
```

```
stringToInt :: String -> Int
stringToInt s = fromMaybe (error "") (readMaybe s)

lastPartInt :: Text.Text -> Int
lastPartInt a = stringToInt (Data.List.last (Data.List.Split.splitOn
";" (Data.Text.unpack a)))
```

```
plusNumber11o1 :: Int -> Int -> Int
plusNumber11o1 a b
 | (b==11) && (a+b>21) = a + 1
 | otherwise = a + b

countScore :: [Text] -> Int
countScore [] = 0
countScore b = Data.List.foldl plusNumber11o1 0 (sort (Data.List.map
lastPartInt b))
```

4.5.

A játékos és a Bank első két kártyáját kiosztjuk.

$playerN \in \mathbb{N}, player \in TCard^{playerN}$

$bankN \in \mathbb{N}, bank \in TCard^{bankN}$

$player_1 = swappedDOC_1$

$bank_1 = swappedDOC_2$

$player_2 = swappedDOC_3$

$bank_2 = swappedDOC_4$

```
static void getACard(ref int handN, ref string[] hand, ref int swap-
pedDOCN, ref string[] swappedDOC) {
 hand[handN] = swappedDOC[0];
 handN++;
 swappedDOCN--;
 for (int i = 0; i < swappedDOCN; i++) {
  swappedDOC[i] = swappedDOC[i + 1];
 }
}

static void showCards(string title, int N, string[] X) {
 Console.Write(title + " : [");
 for (int i = 0; i < N; i++) {
  if (i > 0) {
   Console.Write(",");
  }
  Console.Write("\"" + X[i] + "\"");
 }
 Console.WriteLine("]");
}

int bankN = 0;
string[] bank = new string[N];
int playerN = 0;
string[] player = new string[N];
getACard(ref playerN, ref player, ref swappedDOCN, ref swappedDOC);
getACard(ref bankN, ref bank, ref swappedDOCN, ref swappedDOC);
showCards("You can see this card at Bank", bankN, bank);
Console.WriteLine("  Score: " + countScore(bankN, bank));
getACard(ref playerN, ref player, ref swappedDOCN, ref swappedDOC);
getACard(ref bankN, ref bank, ref swappedDOCN, ref swappedDOC);
```

```
cat .swappedDOC | head -n 1 > .player
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~

cat .swappedDOC | head -n 1 > .bank
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~

echo "You can see this card at Bank : `echo -n '[\"' && ( cat .bank |
tr '\n' ',' | sed 's/,$//' || sed 's/,/\",\"/g' ) && echo -n '\"]'`"
echo "  Score: `countScore .bank`"

cat .swappedDOC | head -n 1 >> .player
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~

cat .swappedDOC | head -n 1 > .bank
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~
```

```
let player = [ (Data.List.head swappedDOC) ]
let swappedDOC2 = Data.List.tail swappedDOC

let bank = [ (Data.List.head swappedDOC2) ]
let swappedDOC3 = Data.List.tail swappedDOC2

putStrLn ("You can see this card at Bank : " ++ show bank)
putStrLn ("  Score: " ++ show (countScore bank))

let player2 = player ++ [ (Data.List.head swappedDOC3) ]
let swappedDOC4 = Data.List.tail swappedDOC3

let bank2 = bank ++ [ (Data.List.head swappedDOC4) ]
let swappedDOC5 = Data.List.tail swappedDOC4
```

4.6.

A játékos kártyahúzásai addig történnek, amíg azt a játékos akarja az addigi kártyáinak pontszámai alapján. Azonban ezt ennyire pontosan nem tudjuk/akarjuk leírni a specifikációban.

$playerN \geq 2 \wedge playerN = \text{`user decision`}$

$\forall i (3 \leq i \leq playerN): player_i = swappedDOC_{2+i}$

```
string val;
do {
 int sz = countScore(playerN, player);
 showCards("Cards of Player", playerN, player);
 Console.WriteLine("  Total score: " + sz);
 if (sz > 21) {
  val = "n";
 } else {
  Console.Write("Do you hit one more card? (y/N):");
  val = Console.ReadLine();
 }
```

```
 if (val == "y") {
  getACard(ref playerN, ref player, ref swappedDOCN, ref swappedDOC);
 }
} while (val == "y");
```

```
while
 sz=`countScore .player`
 echo "Cards of Player :`echo -n '[\"' && ( cat .player | tr '\n' ','
| sed 's/,$//' || sed 's/,/\",\"/g' ) && echo -n '\"]'`"
 echo "  Total score: $sz"
 if [ $sz -gt 21 ]; then
  val="n"
 else
  echo -n "Do you hit one more card? (y/N):"
  read val
 fi
[ "$val" = "y" ]; do
 cat .swappedDOC | head -n 1 >> .player
 mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~
done
```

```
askHitMore :: [Text] -> IO String
askHitMore x = do
 let sc=countScore x
 putStrLn ("Cards of Player: " ++ show x)
 putStrLn ("  Total score: " ++ show sc)
 if (sc > 21) then
  return "n"
 else do
  putStrLn ("Do you hit one more card? (y/N):")
  line <- getLine
  return line

playsPlayer :: ([Text], [Text]) -> ([Text], [Text])
playsPlayer (j, m)
 | (unsafePerformIO ( askHitMore j )) == "y" = playsPlayer ( j ++ [
(Data.List.head m) ] , Data.List.tail m )
 | otherwise = (j,m)

let playerResult=playsPlayer (player2, swappedDOC5)
let player3 = fst playerResult
let swappedDOC6 = snd playerResult
```

4.7.

A Bank kártyahúzásai. A [8] szerint gyakori, hogy a Bank addig húz, amíg minimum el nem éri a 17 értéket.

$$bankN \geq 2$$
$$\forall i (3 \leq i \leq bankN): bank_i = swappedDOC_{playerN+i}$$
$$countScore\big(sortValue(bank_{1..bankN-1})\big) < 17 \ \wedge$$
$$countScore\big(sortValue(bank_{1..bankN})\big) \geq 17$$

```
while (countScore(bankN, bank) < 17) {
 getACard(ref bankN, ref bank, ref swappedDOCN, ref swappedDOC);
}
```

```
while [ `countScore .bank` -lt 17 ]; do
 cat .swappedDOC | head -n 1 >> .bank
 mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~
done
```

```
playsBank :: ([Text.Text], [Text.Text]) -> ([Text.Text], [Text.Text])
playsBank (b, m)
  | (countScore b) < 17  = playsBank ( b ++ [ (Data.List.head m) ] ,
Data.List.tail m )
  | otherwise = (b, m)

let bankResult=playsBank (bank2,swappedDOC6)
let bank3 = fst bankResult
```

4.8.

A kezdő csomag megkeverése úgy történik, hogy többször elvégezzük a következő műveleteket. Véletlenszerűen ketté szedjük a kártyacsomagot, majd az így keletkezett csomagokat összefésüljük úgy, hogy 50 % valószínűséggel vagy az egyikből vagy a másikból választjuk a kártyát.

$$swappedDOC \in TCard^N$$

$$\forall i\big(1 \le i \le Length(X)\big)\exists j\big(1 \le j \le Length(X)\big): swappedDOC_i = deckOfCards_j \wedge$$

$$\forall i\big(1 \le i \le Length(X)\big)\exists j\big(1 \le j \le Length(X)\big): deckOfCards_i = swappedDOC_j$$

```
swappedDOC = File.ReadAllLines("deckOfCards.txt");
int Na;
int Nb;
string[] a = new string[N];
string[] b = new string[N];
Random rand = new Random();
for (int i=1;i<=20;i++) {
 int vel=rand.Next(0,52);
 for (int j=0;j<vel;j++) {
  a[j] = swappedDOC[j];
 }
 Na = vel;
 for (int j=vel;j<N;j++) {
  b[j-vel] = swappedDOC[j];
 }
 Nb = N - Na;

 int ia = 0;
 int ib = 0;
 int ii = 0;
 while ((ia <Na) || (ib <Nb)) {
  if ((ia < Na) && (ib < Nb)) {
   if (rand.Next(0, 100)<50) {
    swappedDOC[ii] = a[ia];
    ii++;
```

```
    ia++;
   } else {
    swappedDOC[ii] = b[ib];
    ii++;
    ib++;
   }
  } else {
   for (int j=ia;j<Na;j++) {
    swappedDOC[ii] = a[ia];
    ii++;
    ia++;
   }
   for (int j = ib; j < Nb; j++) {
    swappedDOC[ii] = b[ib];
    ii++;
    ib++;
   }
  }
 }
}
```

```
file=deckOfCards.txt
cp $file .swappedDOC
echo -n "Cards are swapping"
for i in `seq 20`; do
 echo -n "."
 vel=`expr $RANDOM % 52 + 1`
 #echo $vel
 cat .swappedDOC | head -n $vel > .a
 cnta=`cat .a | wc -l`
 cat .swappedDOC | tail -n +`expr $vel + 1` > .b
 cntb=`cat .b | wc -l`
 ia=1
 ib=1
 echo -n "" > .swappedDOC
 while [ $ia -le $cnta -o $ib -le $cntb ]; do
  if [ $ia -le $cnta -a $ib -le $cntb ]; then
   if [ `expr $RANDOM % 100` -lt 50 ]; then
    cat .a | head -n $ia | tail -n 1 >> .swappedDOC
    ia=`expr $ia + 1`
   else
    cat .b | head -n $ib | tail -n 1 >> .swappedDOC
    ib=`expr $ib + 1`
   fi
  else
   cat .a | tail -n +$ia >> .swappedDOC
   ia=`expr $cnta + 1`
   cat .b | tail -n +$ib >> .swappedDOC
   ib=`expr $cntb + 1`
  fi
 done
done
echo ""
rm .a
rm .b
```

```
randomNumber :: Int -> Int -> Int
randomNumber x y = unsafePerformIO ( getStdRandom ( randomR (x,y) ) )

compose :: [a] -> [a] -> [a]
compose [] []  = []
compose a [] = a
compose [] b = b
compose (a:as) (b:bs) = do
 let vel = randomNumber 1 100
 if vel < 50
  then [a] ++ ( compose as ([b] ++ bs) )
  else [b] ++ ( compose ([a] ++ as) bs )

swapOnce :: [a] -> [a]
swapOnce (x) = do
 let vel = randomNumber 1 52
 let spl = Data.List.splitAt vel x
 compose (fst spl) (snd spl)

swap :: Int -> [a] -> [a]
swap 0 x = x
swap db x = swap ( db - 1 ) ( swapOnce x )

deckOfCards <- fmap Text.lines (Text.readFile "deckOfCards.txt")
let swappedDOC=swap 100 deckOfCards
```

5. Futtatás

A futtatáshoz a [10] linkről letölthető a Windowson futtatható (exe) bináris állományok és a bash script. A kezdő kártyacsomag adatai `szín;figura;érték` formátumban egy fájlban található. A következő táblázatban látható 4 oszlop egymás után 52 sorban:

hearts;2;2	diamonds;2;2	spades;2;2	clubs;2;2
hearts;3;3	diamonds;3;3	spades;3;3	clubs;3;3
hearts;4;4	diamonds;4;4	spades;4;4	clubs;4;4
hearts;5;5	diamonds;5;5	spades;5;5	clubs;5;5
hearts;6;6	diamonds;6;6	spades;6;6	clubs;6;6
hearts;7;7	diamonds;7;7	spades;7;7	clubs;7;7
hearts;8;8	diamonds;8;8	spades;8;8	clubs;8;8
hearts;9;9	diamonds;9;9	spades;9;9	clubs;9;9
hearts;10;10	diamonds;10;10	spades;10;10	clubs;10;10
hearts;J;10	diamonds;J;10	spades;J;10	clubs;J;10
hearts;Q;10	diamonds;Q;10	spades;Q;10	clubs;Q;10
hearts;K;10	diamonds;K;10	spades;K;10	clubs;K;10
hearts;A;11	diamonds;A;11	spades;A;11	clubs;A;11

A következő képeken a három implementáció egy-egy tesztfuttatása látható a fenti sorrendnek megfelelően.

```
Blackjack
You can see this card at Bank : ["spades;7;7"]
  Score: 7
Cards of Player : ["hearts;8;8","clubs;3;3"]
  Total score: 11
Do you hit one more card? (y/N):y
Cards of Player : ["hearts;8;8","clubs;3;3","clubs;4;4"]
  Total score: 15
Do you hit one more card? (y/N):y
Cards of Player : ["hearts;8;8","clubs;3;3","clubs;4;4","hearts;6;6"]
  Total score: 21
Do you hit one more card? (y/N):
Cards of Player : ["hearts;8;8","clubs;3;3","clubs;4;4","hearts;6;6"]
  Total score: 21, final score: 21
Cards of Bank : ["spades;7;7","clubs;A;11"]
  Total score: 18, final score: 18
BLACKJACK Win - 3:2
```

```
Cards are swapping.....................
You can see this card at Bank : ["clubs;J;10"]
  Score: 10
Cards of Player :["hearts;9;9,spades;9;9"]
  Total score: 18
Do you hit one more card? (y/N):
Cards of Player :["hearts;9;9,spades;9;9"]
  Total score: 18, final score: 18
Cards of Bank : ["spades;A;11,hearts;A;11,spades;Q;10"]
  Total score: 22, final score: 0
BUST Win - 2:1
```

```
You can see this card at Bank : ["diamonds;7;7"]
  Score: 7
Cards of Player: ["hearts;8;8","hearts;9;9"]
  Total score: 17
Do you hit one more card? (y/N):
y
Cards of Player: ["hearts;8;8","hearts;9;9","hearts;A;11"]
  Total score: 18
Do you hit one more card? (y/N):
y
Cards of Player: ["hearts;8;8","hearts;9;9","hearts;A;11","clubs;3;3"]
  Total score: 21
Do you hit one more card? (y/N):

Cards of Player : ["hearts;8;8","hearts;9;9","hearts;A;11","clubs;3;3"]
  Total score: 21, final score: 21
Cards of Bank : ["diamonds;7;7","diamonds;6;6","diamonds;K;10"]
  Total score: 23, final score: 0
BLACKJACK Win - 3:2
```

A képeken látható kis formai eltéréseket azért hagytam meg, hogy megkülönböztethető legyen, hogy melyik kép melyik forráskódhoz tartozik, de azért látszik, hogy sikerült lekódolni ugyanazt mindhárom programozási nyelven.

6. Konkluzió

Az itt bemutatott feladat megoldásainak összehasonlítása segítheti a hallgatóinknak a különböző szintaxisok megértését. Észrevehetik, hogy egy feladat több különböző gondolkodásmóddal történő megoldása is ugyanazon eredményre vezethet.

Úgy gondolom, hogy ez a játékot feldolgozó példa elég motiváló lehet, hogy a diákok aktívan részt vegyenek az órákon és minél többen részesülhessenek egy aha-élményben.

Irodalom

1. Programtervező informatikus BSc 2018, Modellező (A) specializáció ajánlott tantervi háló; https://www.inf.elte.hu/dstore/document/1148/Programtervez%C5%91%20informatikus%20BSc%202018,%20Modellez%C5%91%20%28A%29%20mob.%20ablak%20%282023%20j%C3%BAlius%29.pdf; utoljára megtekintve: 2023.10.30.
2. Programtervező informatikus BSc 2018, Szoftvertervező (B) specializáció ajánlott tantervi háló; https://www.inf.elte.hu/dstore/document/1149/Programtervez%C5%91%20informatikus%20BSc%202018,%20Szoftvertervez%C5%91%20%28B%29%20mob.%20ablak%20%282023%20j%C3%BAlius%29.pdf; utoljára megtekintve: 2023.10.30.
3. Programtervező informatikus BSc 2018, Szoftverfejlesztő (C) specializáció ajánlott tantervi háló; https://www.inf.elte.hu/dstore/document/1150/Programtervez%C5%91%20informatikus%20BSc%202018,%20Szoftverfejleszt%C5%91%20%28C%29%20%282023%20j%C3%BAlius%29.pdf; utoljára megtekintve: 2023.10.30.
4. Programozás tematika; https://www.inf.elte.hu/dstore/document/1030/Programoz%C3%A1s.pdf; utoljára megtekintve: 2023.10.30.
5. Számítógépes rendszerek tematika; https://www.inf.elte.hu/dstore/document/1029/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9pes%20rendszerek.pdf; utoljára megtekintve: 2023.10.30.
6. Funkcionális programozás tematika; https://www.inf.elte.hu/dstore/document/1038/Funkcion%C3%A1lis%20programoz%C3%A1s.pdf; utoljára megtekintve: 2023.10.30.
7. Imperatív programozás tematika; https://www.inf.elte.hu/dstore/document/1031/Imperativ%20programoz%C3%A1s.pdf; utoljára megtekintve: 2023.10.30.
8. Blackjack információk; https://en.wikipedia.org/wiki/Blackjack; utoljára megtekintve: 2023.10.30.
9. Blackjack játék implementációim; https://github.com/laszlogmenyhart/blackjack; utoljára megtekintve: 2023.10.30.
10. Első release-ben egy bash script és két Windows-on futtatható bináris (exe); https://github.com/laszlogmenyhart/blackjack/releases/download/v0.1/blackjack-v0.1-bin.zip; utoljára megtekintve: 2023.10.30.

Mellékletek

A. A teljes C# implementáció

```
using System.Collections.Specialized;
using System.Numerics;

namespace play {
 internal class Program {

  static int countScore(int N, string[] arr) {
   int[] values=new int[N];
   for (int i = 0; i < N; i++) {
    values[i]=Int32.Parse(arr[i].Split(";")[2]);
   }
   Array.Sort(values);
   int sum = 0;
   for (int i = 0; i < N; i++) {
    int x = values[i];
    if ((x == 11) && (sum + x > 21)) {
     sum += 1;
    } else {
     sum += x;
    }
   }
   return sum;
  }

  static int finalScore(int score) {
   if (score > 21) {
    return 0;
   } else {
    return score;
   }
  }

  static void showCards(string title,int N,string[] X) {
   Console.Write(title+" : [");
   for (int i = 0; i < N; i++) {
    if (i > 0) {
     Console.Write(",");
    }
    Console.Write("\"" + X[i] + "\"");
   }
   Console.WriteLine("]");
  }

  static void getACard(ref int handN, ref string[] hand, ref int swap-
pedDOCN, ref string[] swappedDOC) {
   hand[handN] = swappedDOC[0];
   handN++;
   swappedDOCN--;
   for (int i = 0; i < swappedDOCN; i++) {
    swappedDOC[i] = swappedDOC[i + 1];
   }
```

```
}

static void Main(string[] args) {
 Console.WriteLine("Blackjack");

 /// declaration
 ///
 int N = 52;
 int swappedDOCN = N;
 string[] swappedDOC;

 string blackjackResult;

 /// read input data
 ///
 swappedDOC = File.ReadAllLines("deckOfCards.txt");

 /// solve the problem - implement algorithms
 ///
 int Na;
 int Nb;
 string[] a = new string[N];
 string[] b = new string[N];
 Random rand = new Random();
 for (int i=1;i<=20;i++) {
  int vel=rand.Next(0,52);
  for (int j=0;j<vel;j++) {
   a[j] = swappedDOC[j];
  }
  Na = vel;
  for (int j=vel;j<N;j++) {
   b[j-vel] = swappedDOC[j];
  }
  Nb = N - Na;

  int ia = 0;
  int ib = 0;
  int ii = 0;
  while ((ia <Na) || (ib <Nb)) {
   if ((ia < Na) && (ib < Nb)) {
    if (rand.Next(0, 100)<50) {
     swappedDOC[ii] = a[ia];
     ii++;
     ia++;
    } else {
     swappedDOC[ii] = b[ib];
     ii++;
     ib++;
    }
   } else {
    for (int j=ia;j<Na;j++) {
     swappedDOC[ii] = a[ia];
     ii++;
     ia++;
    }
```

```
      for (int j = ib; j < Nb; j++) {
       swappedDOC[ii] = b[ib];
       ii++;
       ib++;
      }
     }
    }
   }
   // showCards("Swapped cards", swappedDOCN, swappedDOC);

   int bankN = 0;
   string[] bank = new string[N];
   int playerN = 0;
   string[] player = new string[N];

   getACard(ref playerN, ref player, ref swappedDOCN, ref swappedDOC);
   getACard(ref bankN, ref bank, ref swappedDOCN, ref swappedDOC);
   showCards("You can see this card at Bank", bankN, bank);
   Console.WriteLine("  Score: " + countScore(bankN, bank));
   getACard(ref playerN, ref player, ref swappedDOCN, ref swappedDOC);
   getACard(ref bankN, ref bank, ref swappedDOCN, ref swappedDOC);

   string val;
   do {
    int sz = countScore(playerN, player);
    showCards("Cards of Player", playerN, player);
    Console.WriteLine("  Total score: " + sz);
    if (sz > 21) {
     val = "n";
    } else {
     Console.Write("Do you hit one more card? (y/N):");
     val = Console.ReadLine();
    }
    if (val == "y") {
     getACard(ref playerN,ref player,ref swappedDOCN,ref swappedDOC);
    }
   } while (val == "y");

   int playerScore = countScore(playerN, player);
   int playerScoreFinal = finalScore(playerScore);

   while (countScore(bankN, bank) <= 16) {
    getACard(ref bankN, ref bank, ref swappedDOCN, ref swappedDOC);
   }
   int bankScore = countScore(bankN, bank);
   int bankScoreFinal = finalScore(bankScore);

   if ((playerScoreFinal == 21) && (bankScoreFinal < 21)) {
    blackjackResult = "BLACKJACK Win - 3:2";
   } else if ((playerScoreFinal == bankScoreFinal) && (playerScoreFinal
> 0)) {
    blackjackResult = "PUSH Get back - 1:1";
   } else if (playerScoreFinal > bankScoreFinal) {
    blackjackResult = "BUST Win - 2:1";
   } else {
```

```
    blackjackResult = "YOU LOST - 0:1";
   }

   /// write out answers
   ///
   showCards("Cards of Player", playerN, player);
   Console.WriteLine("  Total score: " + playerScore + ", final score: "
+ playerScoreFinal);

   showCards("Cards of Bank", bankN,bank);
   Console.WriteLine("  Total score: " + bankScore + ", final score: " +
bankScoreFinal);
   Console.WriteLine(blackjackResult);
  }
 }
}
```

B. A teljes Bash implementáció

```
#!/bin/bash

function countScore() {
 sum=0
 for x in `cat $1 | cut -f 3 -d";" | sort -n`; do
  if [ $x -eq 11 -a `expr $sum + $x` -gt 21 ]; then
   sum=`expr $sum + 1`
  else
   sum=`expr $sum + $x`
  fi
 done
 echo $sum
}

function finalScore() {
 if [ $1 -gt 21 ]; then
  echo 0
 else
  echo $1
 fi
}

#file=oneSuiteOfCards.txt
file=deckOfCards.txt
cp $file .swappedDOC
echo -n "Cards are swapping"
for i in `seq 20`; do
 echo -n "."
 vel=`expr $RANDOM % 52 + 1`
 #echo $vel
 cat .swappedDOC | head -n $vel > .a
 cnta=`cat .a | wc -l`
 cat .swappedDOC | tail -n +`expr $vel + 1` > .b
 cntb=`cat .b | wc -l`
 ia=1
 ib=1
 echo -n "" > .swappedDOC
```

```
 while [ $ia -le $cnta -o $ib -le $cntb ]; do
  if [ $ia -le $cnta -a $ib -le $cntb ]; then
   if [ `expr $RANDOM % 100` -lt 50 ]; then
    cat .a | head -n $ia | tail -n 1 >> .swappedDOC
    ia=`expr $ia + 1`
   else
    cat .b | head -n $ib | tail -n 1 >> .swappedDOC
    ib=`expr $ib + 1`
   fi
  else
   cat .a | tail -n +$ia >> .swappedDOC
   ia=`expr $cnta + 1`
   cat .b | tail -n +$ib >> .swappedDOC
   ib=`expr $cntb + 1`
  fi
 done
done
echo ""
rm .a
rm .b
#cat .swappedDOC

cat .swappedDOC | head -n 1 > .player
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~

cat .swappedDOC | head -n 1 > .bank
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~

echo "You can see this card at Bank : `echo -n '[\"' && ( cat .bank | tr
'\n' ',' | sed 's/,$//' || sed 's/,/\",\"/g' ) && echo -n '\"]'`"
echo "  Score: `countScore .bank`"

cat .swappedDOC | head -n 1 >> .player
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~

cat .swappedDOC | head -n 1 > .bank
mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~

while
 sz=`countScore .player`
 echo "Cards of Player :`echo -n '[\"' && ( cat .player | tr '\n' ',' |
sed 's/,$//' || sed 's/,/\",\"/g' ) && echo -n '\"]'`"
 echo "  Total score: $sz"
 if [ $sz -gt 21 ]; then
  val="n"
 else
  echo -n "Do you hit one more card? (y/N):"
  read val
 fi
[ "$val" = "y" ]; do
 cat .swappedDOC | head -n 1 >> .player
```

```
 mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~
done
playerScore=`countScore .player`
playerScoreFinal=`finalScore $playerScore`

while [ `countScore .bank` -le 16 ]; do
 cat .swappedDOC | head -n 1 >> .bank
 mv .swappedDOC .swappedDOC~ && cat .swappedDOC~ | tail -n +2 >> .swap-
pedDOC && rm .swappedDOC~
done
bankScore=`countScore .bank`
bankScoreFinal=`finalScore $bankScore`

if [ $playerScoreFinal -eq 21 -a $bankScoreFinal -lt 21 ]; then
 blackjackResult="BLACKJACK Win - 3:2"
elif [ $playerScoreFinal -eq $bankScoreFinal -a $playerScoreFinal -gt 0
]; then
 blackjackResult="PUSH Get back - 1:1"
elif [ $playerScoreFinal -gt $bankScoreFinal ]; then
 blackjackResult="BUST Win - 2:1"
else
 blackjackResult="YOU LOST - 0:1"
fi

echo "Cards of Player :`echo -n '[\"' && ( cat .player | tr '\n' ',' |
sed 's/,$//' || sed 's/,/\",\"/g' ) && echo -n '\"]'`"
echo "  Total score: $playerScore, final score: $playerScoreFinal"
echo "Cards of Bank : `echo -n '[\"' && ( cat .bank | tr '\n' ',' | sed
's/,$//' || sed 's/,/\",\"/g' ) && echo -n '\"]'`"
echo "  Total score: $bankScore, final score: $bankScoreFinal"
echo $blackjackResult

rm .swappedDOC
rm .player
rm .bank
```

C. A teljes Haskell implementáció

```
import qualified Data.Text as Text
import qualified Data.Text.IO as Text
import System.Random
import System.IO.Unsafe
import Data.List
import Data.List.Split
import Data.Text
import Text.Read (readMaybe)
import Data.Maybe (fromMaybe)


stringToInt :: String -> Int
stringToInt s = fromMaybe (error "") (readMaybe s)

compose :: [a] -> [a] -> [a]
compose [] []  = []
compose a [] = a
compose [] b = b
compose (a:as) (b:bs) = do
```

```
 let vel = randomNumber 1 100
 if vel < 50
  then [a] ++ ( compose as ([b] ++ bs) )
  else [b] ++ ( compose ([a] ++ as) bs )

swapOnce :: [a] -> [a]
swapOnce (x) = do
 let vel = randomNumber 1 52
 let spl = Data.List.splitAt vel x
 compose (fst spl) (snd spl)

swap :: Int -> [a] -> [a]
swap 0 x = x
swap db x = swap ( db - 1 ) ( swapOnce x )

randomNumber :: Int -> Int -> Int
randomNumber x y = unsafePerformIO ( getStdRandom ( randomR (x,y) ) )

askHitMore :: [Text] -> IO String
askHitMore x = do
 let sc=countScore x
 putStrLn ("Cards of Player: " ++ show x)
 putStrLn ("  Total score: " ++ show sc)
 if (sc > 21) then
  return "n"
 else do
  putStrLn ("Do you hit one more card? (y/N):")
  line <- getLine
  return line

playsPlayer :: ([Text], [Text]) -> ([Text], [Text])
playsPlayer (j, m)
 | (unsafePerformIO ( askHitMore j )) == "y" = playsPlayer ( j ++ [ (Data.List.head
m) ] , Data.List.tail m )
 | otherwise = (j,m)

lastPartInt :: Text.Text -> Int
lastPartInt  a  =  stringToInt  (Data.List.last  (Data.List.Split.splitOn  ";"
(Data.Text.unpack a)))

plusNumber11o1 :: Int -> Int -> Int
plusNumber11o1 a b
 | (b==11) && (a+b>21) = a + 1
 | otherwise = a + b

playsBank :: ([Text.Text], [Text.Text]) -> ([Text.Text], [Text.Text])
playsBank (b, m)
  | (countScore b) <= 16  = playsBank ( b ++ [ (Data.List.head m) ] , Data.List.tail
m )
  | otherwise = (b, m)

countScore :: [Text] -> Int
countScore [] = 0
countScore b = Data.List.foldl plusNumber11o1 0 (sort (Data.List.map lastPartInt
b))

finalScore :: Int -> Int
finalScore x
 | x > 21 = 0
 | otherwise = x

main :: IO()
```

```
main = do
 deckOfCards <- fmap Text.lines (Text.readFile "deckOfCards.txt")

 let swappedDOC=swap 100 deckOfCards
 {-}
 print $ swappedDOC
 -}

 let player = [ (Data.List.head swappedDOC) ]
 let swappedDOC2 = Data.List.tail swappedDOC

 let bank = [ (Data.List.head swappedDOC2) ]
 let swappedDOC3 = Data.List.tail swappedDOC2

 putStrLn ("You can see this card at Bank : " ++ show bank)
 putStrLn ("  Score: " ++ show (countScore bank))

 let player2 = player ++ [ (Data.List.head swappedDOC3) ]
 let swappedDOC4 = Data.List.tail swappedDOC3

 let bank2 = bank ++ [ (Data.List.head swappedDOC4) ]
 let swappedDOC5 = Data.List.tail swappedDOC4

 let playerResult=playsPlayer (player2, swappedDOC5)
 let player3 = fst playerResult
 let swappedDOC6 = snd playerResult
 let playerScore=countScore player3
 let playerScoreFinal = finalScore playerScore

 let bankResult=playsBank (bank2,swappedDOC6)
 let bank3 = fst bankResult
 let bankScore=countScore bank3
 let bankScoreFinal = finalScore bankScore

 let blackjackResult = if ((playerScoreFinal == 21) && (bankScoreFinal < 21))
  then "BLACKJACK Win - 3:2"
  else if ((playerScoreFinal == bankScoreFinal) && (playerScoreFinal > 0))
  then "PUSH Get back - 1:1"
  else if (playerScoreFinal > bankScoreFinal)
  then "BUST Win - 2:1"
  else "YOU LOST - 0:1"

 putStrLn ("Cards of Player : " ++ show player3)
 putStrLn ("  Total score: " ++ show playerScore ++ ", final score: " ++ show
playerScoreFinal)
 putStrLn ("Cards of Bank : " ++ show bank3)
 putStrLn ("  Total score: " ++ show bankScore ++ ", final score: " ++ show
bankScoreFinal)
 putStrLn (blackjackResult)
```