Eötvös Lóránd Tudományegyetem
Informatikai Kar
Algoritmusok és Alkalmazásaik Tanszék

Struktogramkészítő és kódgeneráló program

Témavezető:
dr. Ásványi Tibor
egyetemi docens

Készítette:
Békefi Bianka Flóra
programtervező informatikus BSc

Budapest, 2016

Tartalomjegyzék

Bevezetés

A szakdolgozat – címének megfelelően – struktogramok készítésére és azokból való kódgenerálásra szolgál.

Először érdemes definiálni, mit nevezünk struktogramnak. A struktogramok algoritmusok leírására és grafikus megjelenítésére szolgálnak, építőelemeik a szekvencia, elágazás, valamint a ciklusok (elöltesztelős, hátultesztelős, számlálós). A programozási nyelveknél kevésbé kötött a szintaktikájuk.

A szakdolgozatom célja egy olyan program, mellyel kényelmesen és gyorsan készíthetőek struktogramok. Ezek aztán exportálhatóak képként, így dokumentumokba, algoritmusok leírásához, szemléltetéséhez használhatóak. Másrészt képes C++ forráskód generálására az algoritmusból, mely kisebb kiegészítésekkel (pl.: könyvtárak importálása) futtatható lesz. A struktogramokban használhatóak a megszokott logikai kifejezések (and, or, stb.), a kódgenerálás során ezek kicserélődnek a nyelv megfelelő kifejezéseire.

1. Felhasználói dokumentáció

A feladat egy olyan program készítése volt, mellyel kényelmesen készíthetőek tetszőlegesen komplex struktogramok, és amely képes ezekből olyan forráskódot generálni, ami kisebb kiegészítésekkel futtatható.

1.1. Rendszerkövetelmények

A program Windows 8.1 operációs rendszeren készült, .NET Framework 4 keretrendszerben. A futtatáshoz szükséges Windows operációs rendszer. .NET nem kell, mivel a program telepítése során a szükséges .NET keretrendszer is települ a számítógépre.

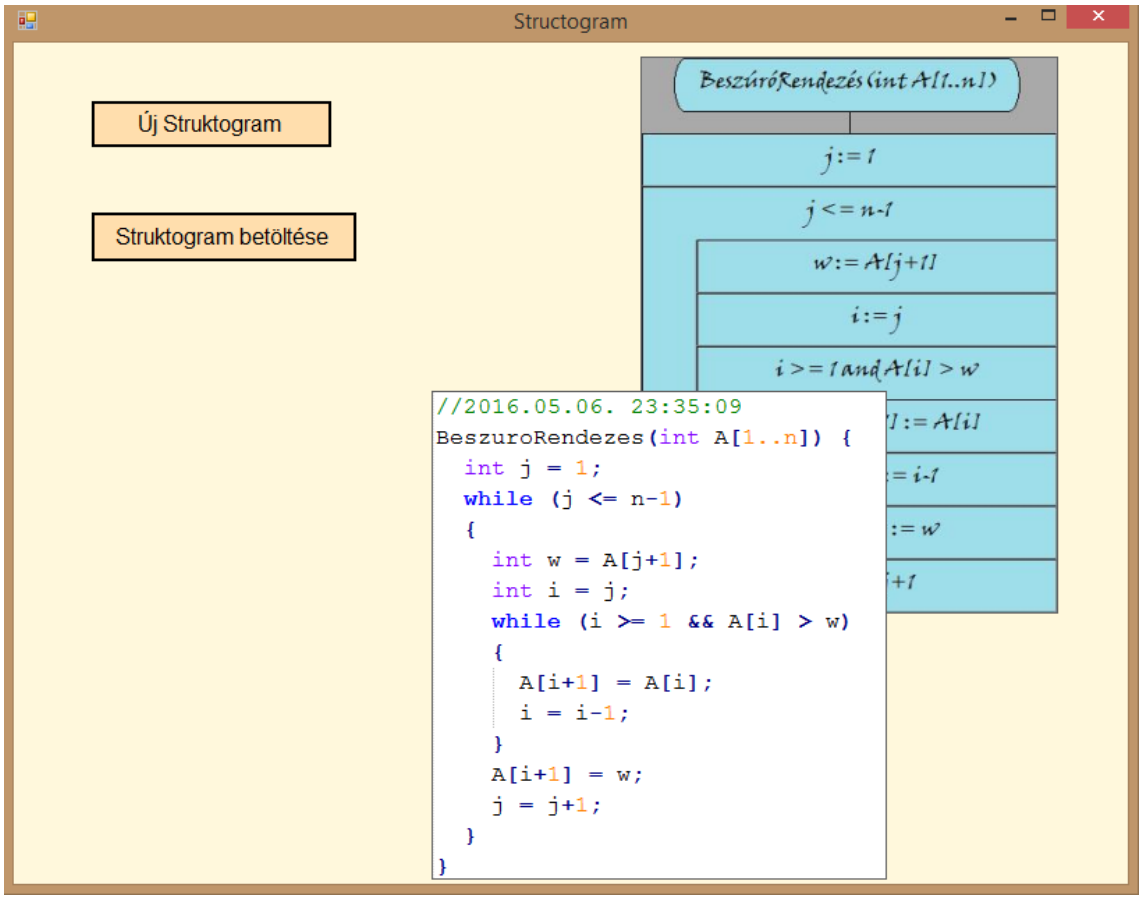
1.2. Telepítés

Az alkalmazást az első használat előtt telepíteni kell. Ehhez a mellékelt lemezen a „telepites” mappában található setup.exe fájlt kell elindítani. Ezután a program magától elindul. A későbbiekben az alkalmazás elindítható mind a setup.exe, mind a Structogram.application fájlok futtatásával. Ha a számítógépen Avast vírusirtó van, akkor azt ki kell kapcsolni a telepítés, illetve az alkalmazás futtatása előtt.

1.3. A program használata

1.3.1. Üzembe helyezés

A Structogram.exe fájlt elindítva elindul a program. Ez a kép fogad minket, a funkciók részletes leírása a következő fejezetben található.

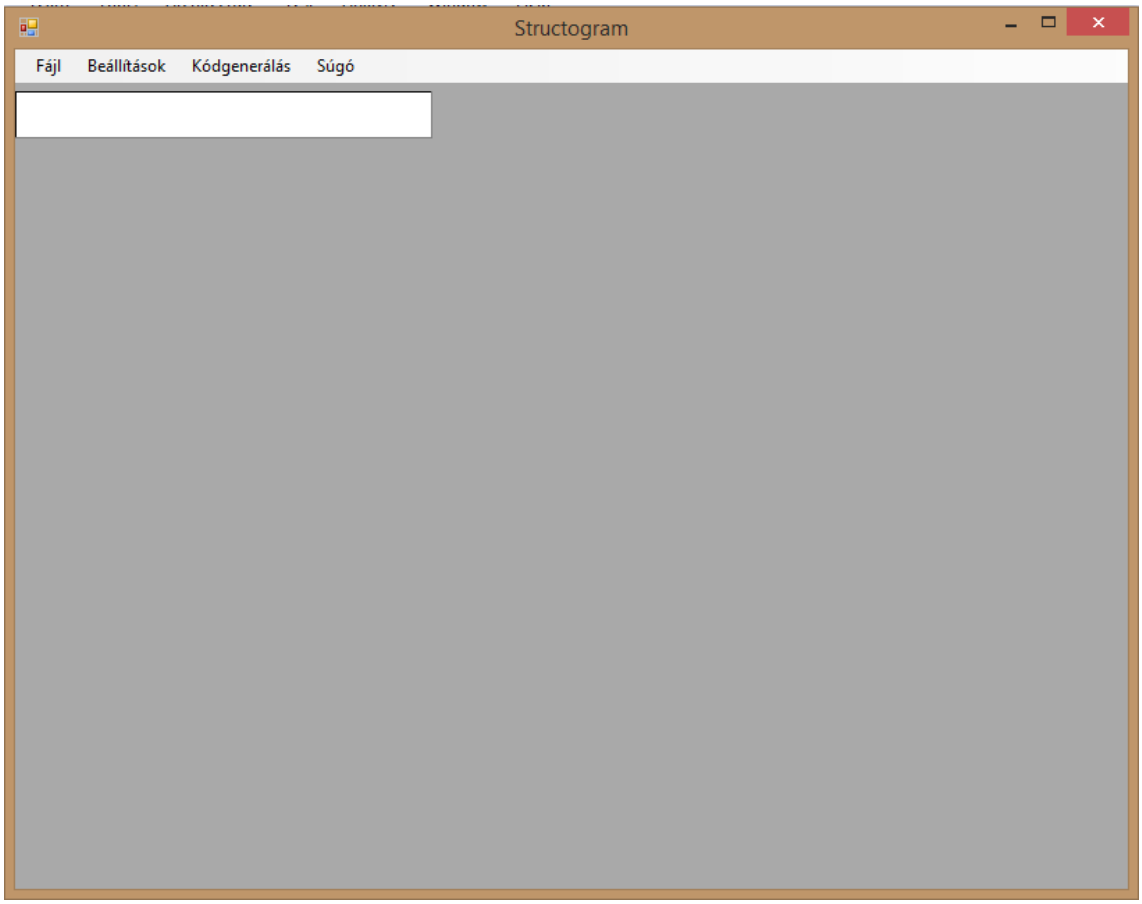


Ábra 1

1.3.2. Funkciók

Az alkalmazásban lehetőség van új struktogram készítésére, struktogramok szerkesztésére, mentésére, kódgenerálásra belőlük. Be lehet tölteni korábbi struktogramot, továbbá beállítható a kinézetük (háttérszín, betűtípus, stb.).

1.3.3. Új struktogram



Ábra 2

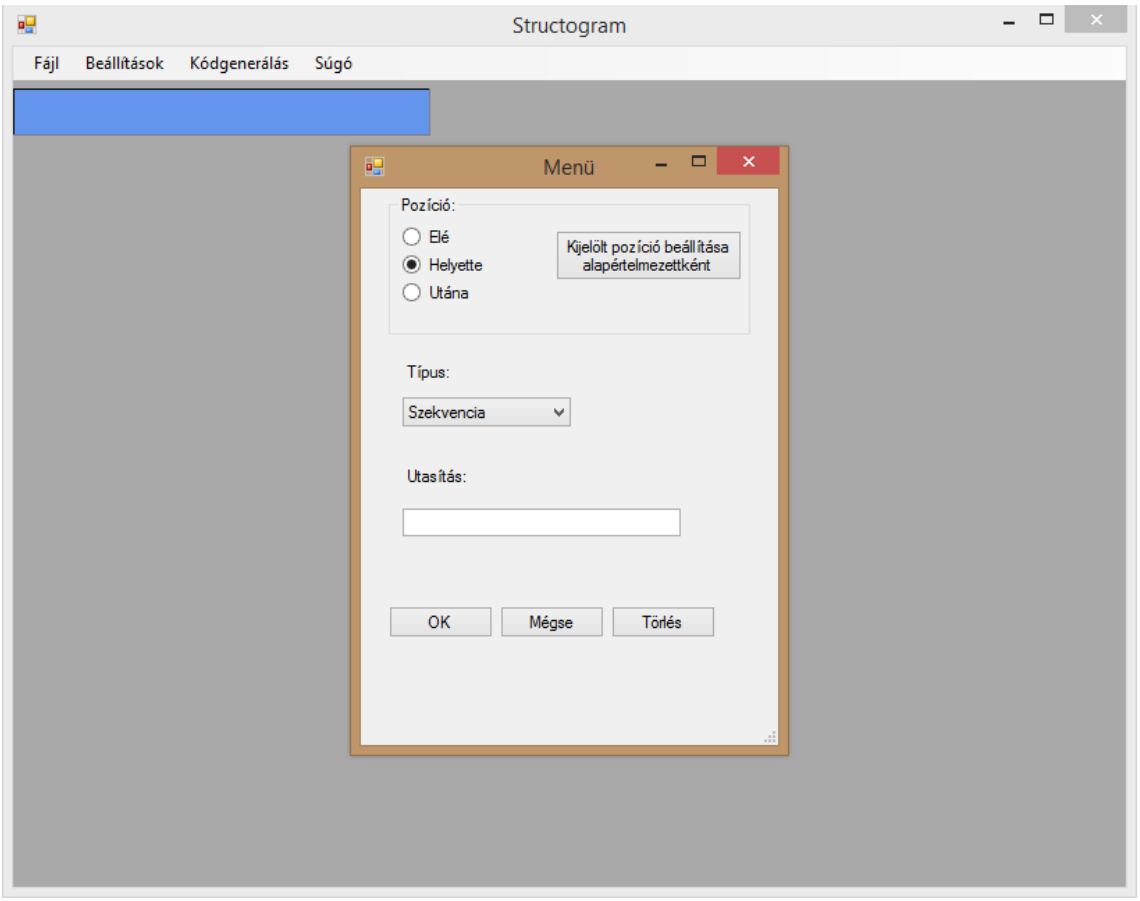
A program elindítása után a kezdőmenüben két menüpont közül választhatunk. Új struktogram készítése, vagy már létező struktogram betöltése. Az „Új Struktogram” menüpontot kiválasztva egy üres téglalap jelenik meg, egy szekvencia. Erre kattintva tudunk módosításokat végezni rajta, új elemeket hozzáadni.

A későbbiekben is bármikor kezdhetünk új struktogramot, ekkor az alkalmazás figyelmeztet rá, ha a jelenlegit még nem mentettük el.

1.3.4. Struktogram szerkesztése

A struktogramok szerkesztése során a következő lehetőségek közül választhatunk:

- Új elem beszúrása a jelenlegi elé vagy után
- Az aktuális elem módosítása
- Az aktuális elem törlése



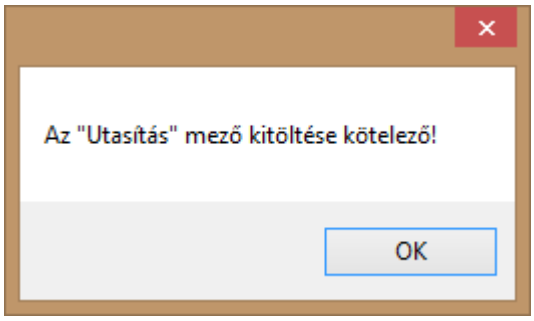
Ábra 3

Ha már elkezdtünk rajzolni egy struktogramot, az egyes elemekre kattintva megjelenik egy menü. Az az elem, amelyre kattintottunk, átszíneződik, hogy a felhasználó tudja, melyiket választotta ki.

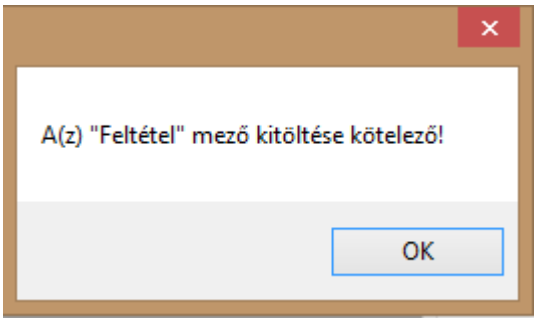
A menüben megadhatjuk, hogy az aktuális elemet szeretnénk módosítani, vagy pedig elé, illetve utána szeretnénk beszúrni egy elemet. A pozíciónál a kiválasztott értéket beállíthatjuk alapértelmezettként, így például, ha új struktogramot rakunk össze, akkor az „Utána” értékre beállítva, nem kell minden egyes alkalommal újra kiválasztanunk. A pozíció alapértelmezett értéke a „Helyette”, amíg át nem állítjuk.

Az elemek típusa lehet szekvencia, elágazás, elöltesztelős, hátultesztelős, illetve számlálós ciklus. Alapértelmezett típusa megegyezik azzal a típussal, amelyre rákattintva megnyílt a menü.

Meg kell még adni ezen kívül, hogy milyen utasítást, vagy feltételt akarunk beleírni a struktogram adott elemébe. Ennek a mezőnek a kitöltése kötelező, ha mégis elmulasztjuk, akkor egy felugró ablak figyelmeztet minket.

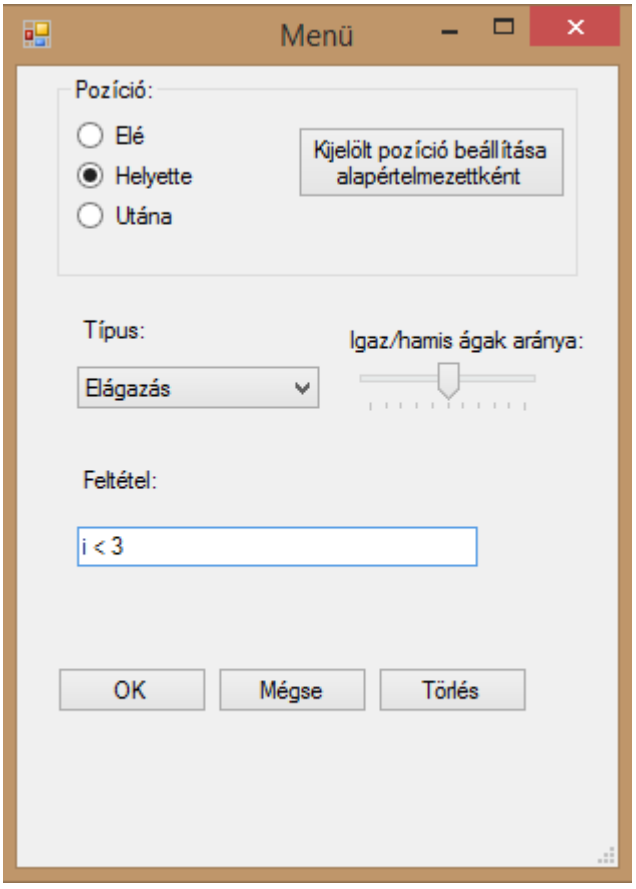


Ábra 4



Ábra 5

Ha a kiválasztott típus az elágazás, akkor a menü módosul, ki kell választani az igaz és a hamis ágak szélességének arányát. Ennek az alapértelmezett értéke 50-50 %.



Ábra 6

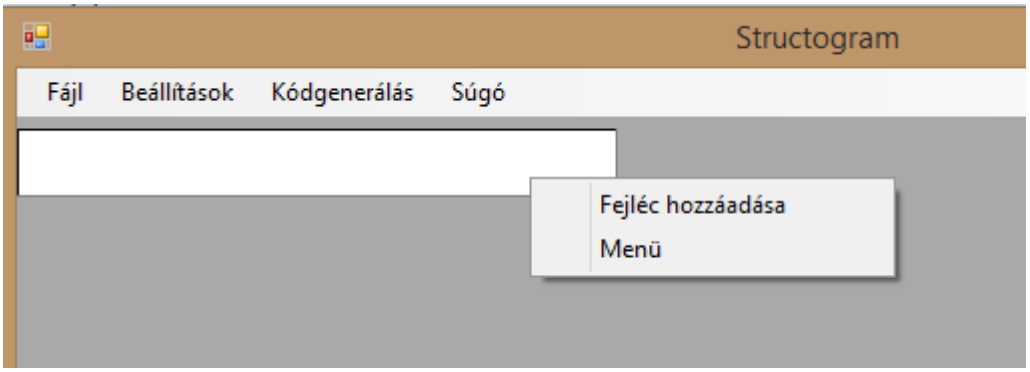
Ha a kiválasztott típus a számlálós ciklus, akkor az utasítás mezőbe kattintva megjelenik egy súgó. Ez megmutatja, hogy a kódgeneráláshoz milyen szintaktikában kell megadni a ciklus feltételét (pl.: i:=1..5, vagy i:=10..0 (-2)). Ettől el lehet térni, de akkor a generált kód nem lesz futtatható, a felhasználónak ki kell majd javítania a kódot.

Az OK gombra kattintva végrehajtódnak a kiválasztott változtatások. Amennyiben nem adjuk meg a megjelenítendő utasítást, egy felugró ablak figyelmeztet rá, hogy ennek a mezőnek a kitöltése kötelező. Számlálós ciklus esetén pedig, ha nem megfelelő a feltétel szintaktikája, arra is figyelmeztet.

A Törlés gomb hatására kitörlődik az az elem, amelyre kattintva megnyitottuk a menüt. Ha törölni szeretnénk valamit, akkor nem szükséges a menüben semmit sem beállítani, csak a Törlés gombra kell kattintani. Ha elágazást törlünk, akkor az igaz és a hamis ága is törlődik, ciklus esetén pedig a teljes ciklusmag is.

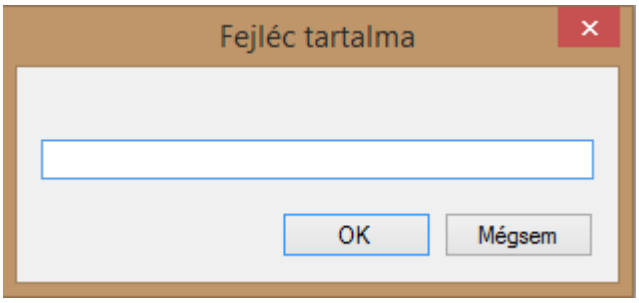
1.3.5. Fejléc hozzáadása, törlése

Alapértelmezetten nincs fejléce a struktogramnak, de jobb gombbal való kattintásra előjön egy menü. Itt kiválaszthatjuk, hogy az előző menüt szeretnénk-e megnyitni (ahol elemek módosíthatóak, új elem hozzáadható), vagy pedig fejlécet szeretnénk hozzáadni.



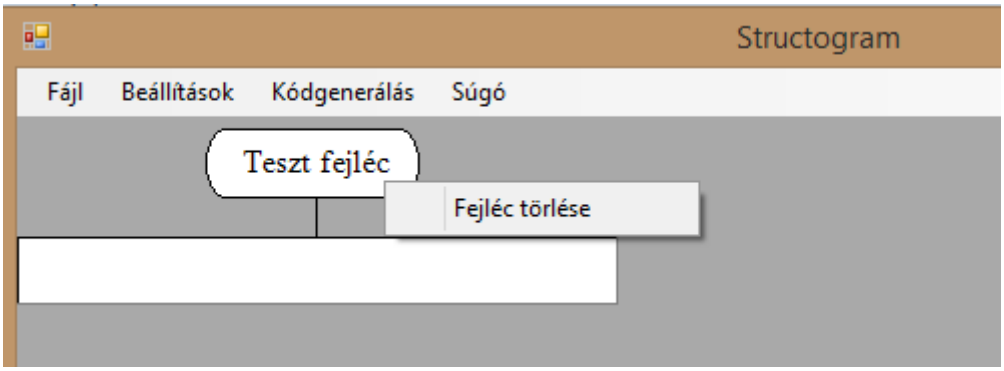
Ábra 7

A Fejléc hozzáadása menüpontot kiválasztva megjelenik egy oválisban a függvény fejléce. Erre rákattintva beírhatjuk a megjelenő ablakba, hogy mi legyen a függvény fejlécének a szövege. Ez szolgál a függvény szignatúrájaként a generált kódban.



Ábra 8

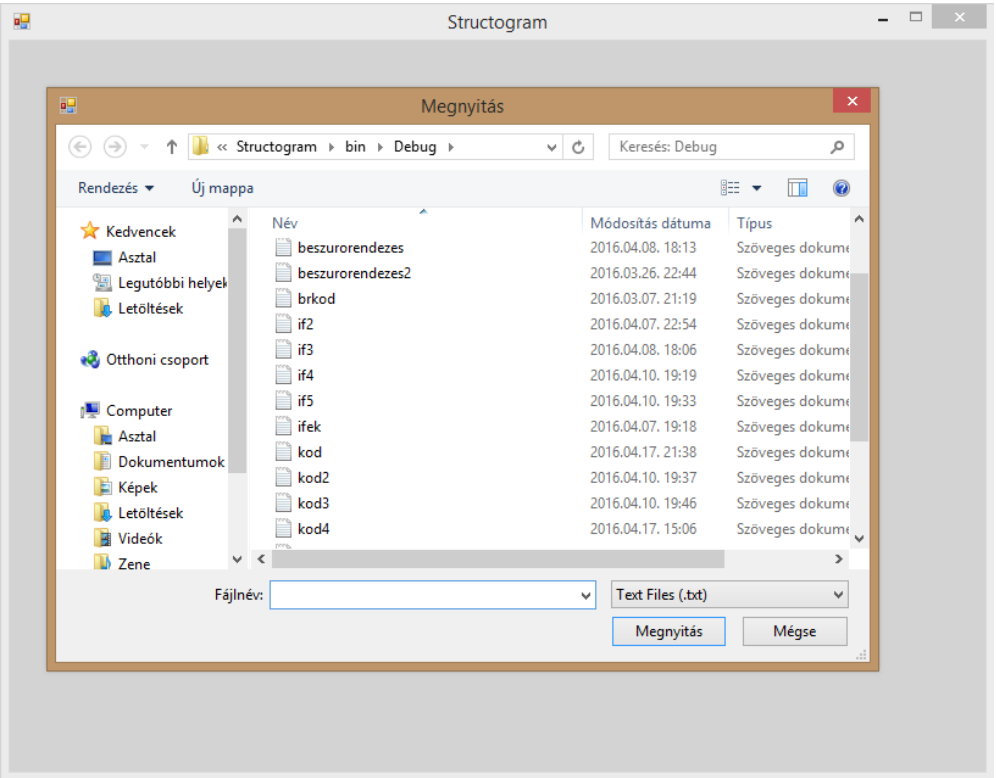
Ha már hozzáadtunk fejlécet a struktogramhoz, akkor a jobb gombra megjelenő menü a Fejléc hozzáadása helyett a Fejléc törlése opciót tartalmazza. Ezt kiválasztva törlődik a fejléc.



Ábra 9

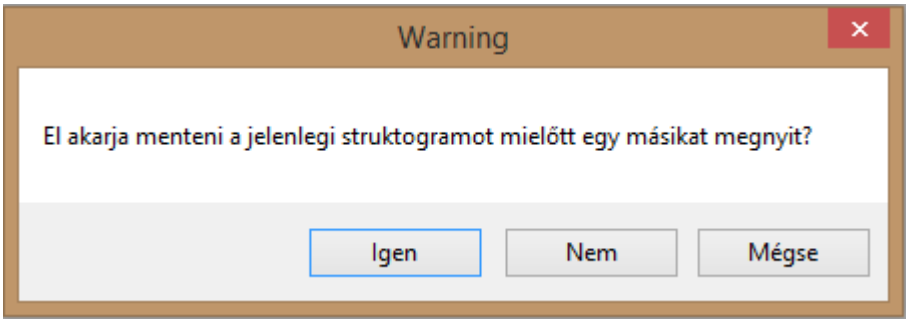
1.3.6. Struktogram betöltése

A „Struktogram betöltése” menüpontra kattintva ki kell választani, hogy melyik fájlt szeretnénk betölteni.



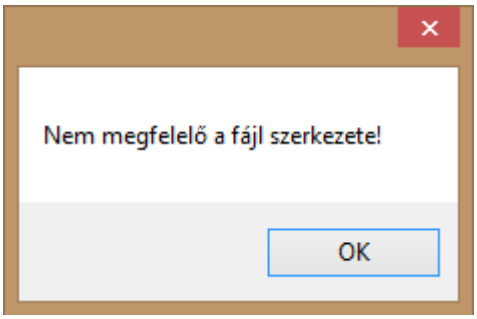
Ábra 10

Ha úgy akarunk új struktogramot betölteni, hogy a jelenlegit még nem mentettük el, akkor egy felugró ablakban erre figyelmeztet.



Ábra 11

Amennyiben nem a program által elvárt struktúrájú fájlt próbálunk meg betölteni, megjelenik egy hibaüzenet, ami közli, hogy nem megfelelő a fájl szerkezete.

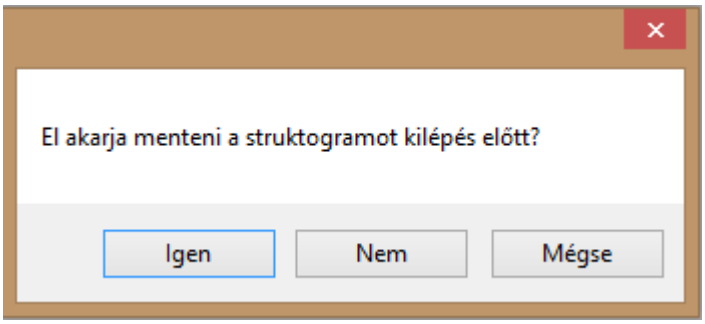


Ábra 12

1.3.7. Struktogram mentése

A Mentés menüpontra kattintva megjelenik egy ablak, ahol ki kell választani a fájl nevét és a könyvtárat, ahova menteni akarja a struktogramot. A fájl kiterjesztése txt lesz. Ez a menüpont a struktogram szerkezetét menti el, nem a kinézetét.

A program bezárásakor, ha még nem mentettük el a struktogramot, akkor kapunk erről egy figyelmeztetést. Az el nem mentett struktogram elveszik.



Ábra 13

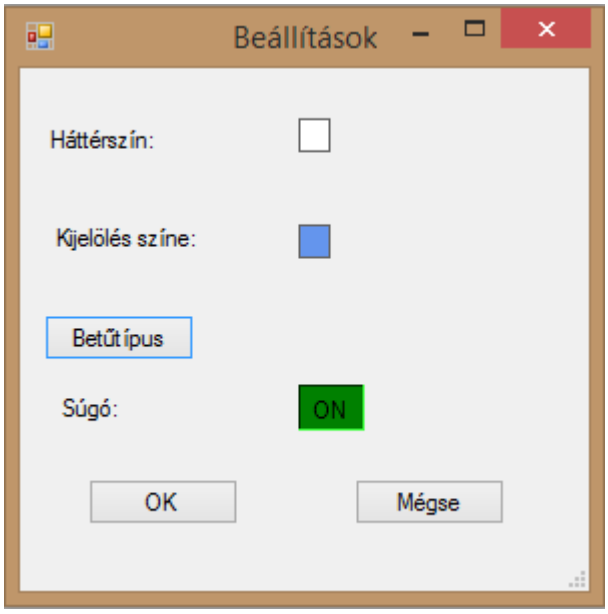
1.3.8. Struktogram exportálása képként

Az Exportálás JPG-ként menüpont ugyanúgy működik, mint a Mentés, csak itt a fájl kiterjesztése jpg lesz és ekkor nem a struktogram szerkezete mentődik el, hanem képként, ahogyan a felhasználó látja.

1.3.9. Kódgenerálás

A Kódgenerálás menüpontot kiválasztva megjelenik a megszokott fájlkiválasztó ablak, ahol meg kell adni a fájl nevét és a mentés helyét. Ezután a program elkészíti a C++ forráskódot a struktogramból.

1.3.10. Beállítások



Ábra 14

A Beállítások menüpontra kattintva megnyílik egy új ablak. Itt lehet beállítani a struktogram háttérszínét, a kijelölés színét (ha a felhasználó rákattint valamelyik elemre, az milyen színű legyen), a betűtípust és itt lehet be-, illetve kikapcsolni a súgót.

1.3.11. Súgó

A „Súgó” menüpontot kiválasztva megjelenik egy használati útmutató a programhoz. Ez tartalmazza azt a szintaktikát, melynek betartása elősegíti a struktogramból generált forráskód helyességét.

A súgó az alkalmazás használata közben is megjelenhet. Ha a felhasználó a kurzort a struktogram egyes elemeire húzza, akkor annak a típusától függően különböző segítséget kaphat. Pl.: milyen legyen a számlálós ciklus feltételének a szintaktikája, hogyan lehet fejlécet hozzáadni, stb. Ez a funkció a beállítások menüpontban be-, illetve kikapcsolható.

2. Fejlesztői dokumentáció

2.1. Követelményanalízis

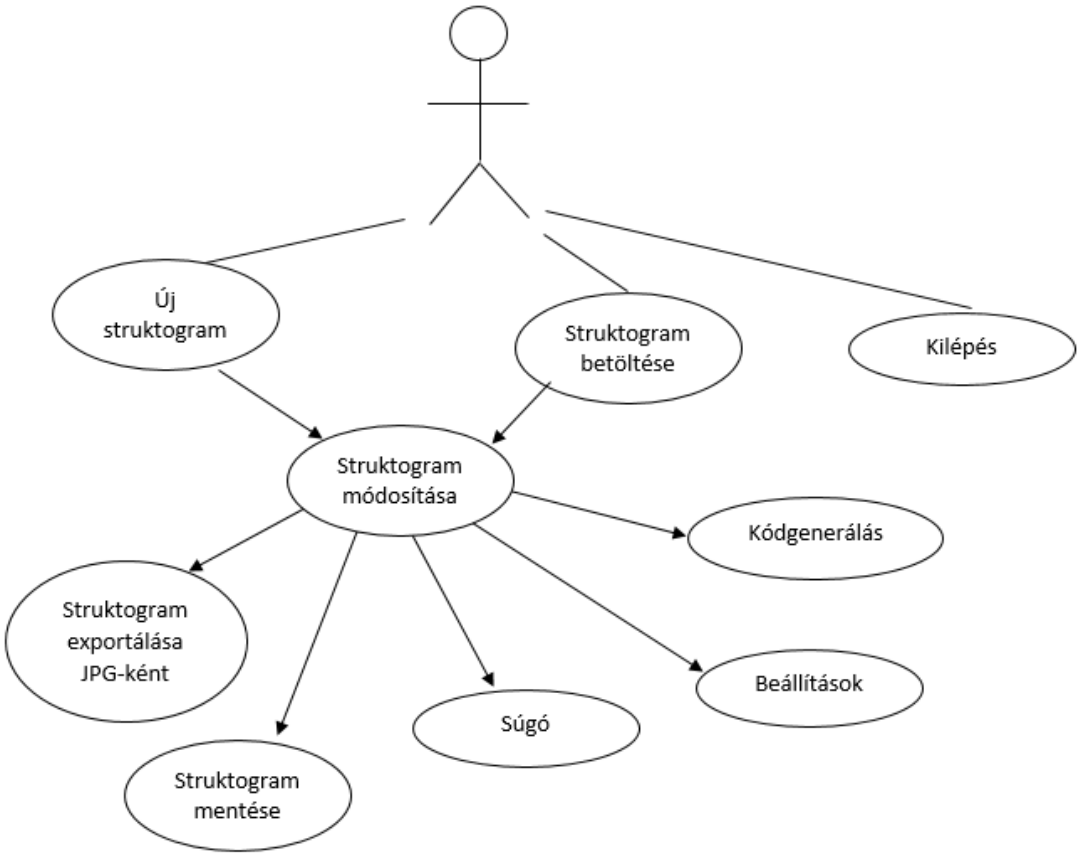
2.1.1. Funkcionális elvárások

- Új struktogram készítése
- Korábbi struktogram betöltése
- Struktogram exportálása képként
- Struktogramok kinézetének beállítása
- Beállítások mentése
- Kódgenerálás struktogramból

2.1.2. Nem funkcionális követelmények

- Használhatóság: a felhasználók számára a felület legyen jól átlátható, könnyen használható
- Biztonság: az elmentett adatok ne vesszenek el, tudja kezelni a hibás bemeneteket, figyelmeztesse a felhasználót, ha szükséges
- Karbantarthatóság: a program legyen könnyen bővíthető új funkciókkal, új elemekkel a struktogramhoz, további kódgenerálási szabályokkal

2.1.3. Felhasználói esetek

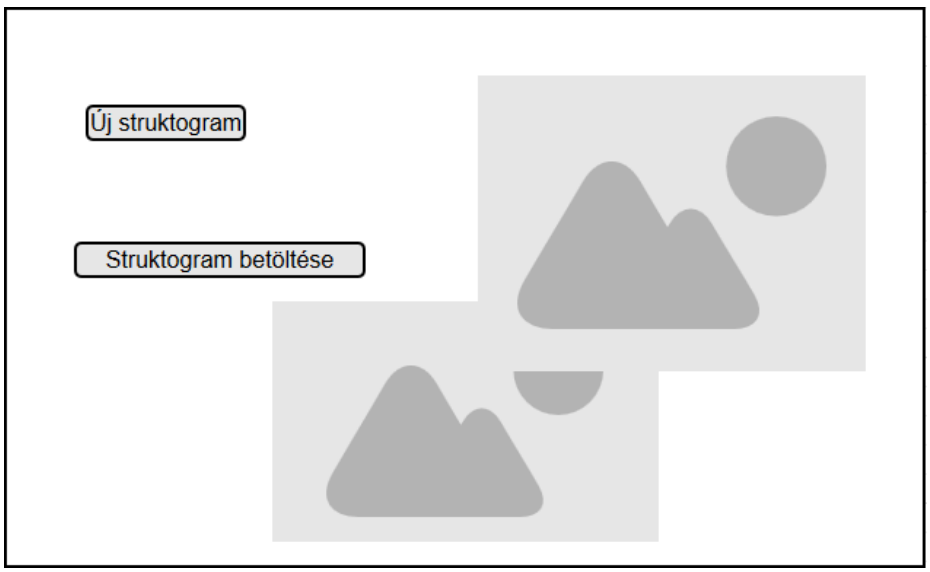


Ábra 15

2.2. Felhasználói felület terve

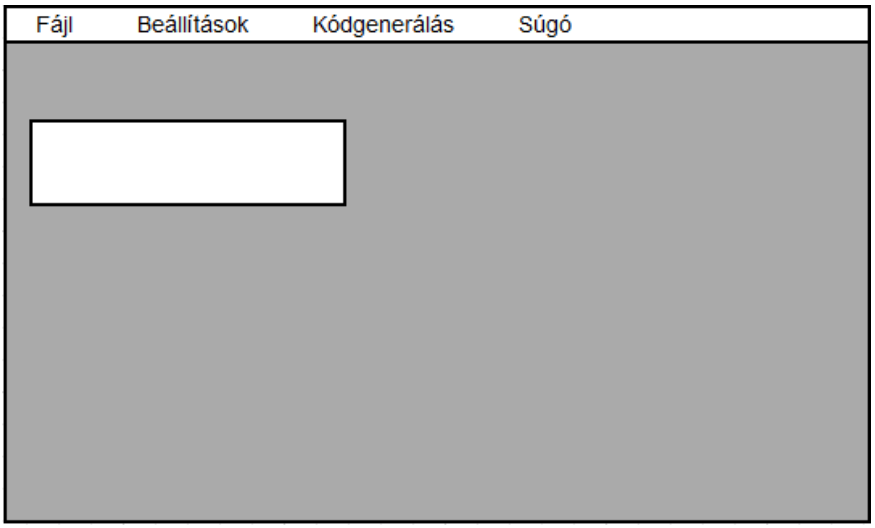
2.2.1. Képernyőtervek

A kezdőképernyő terve:



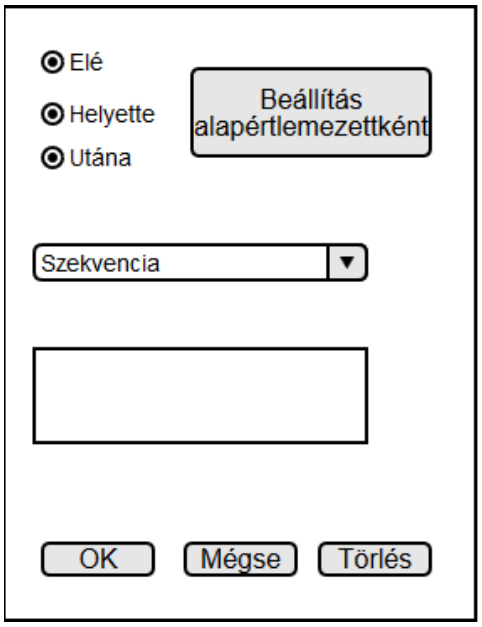
Ábra 16

Az struktogramot megjelenítendő képernyő terve:



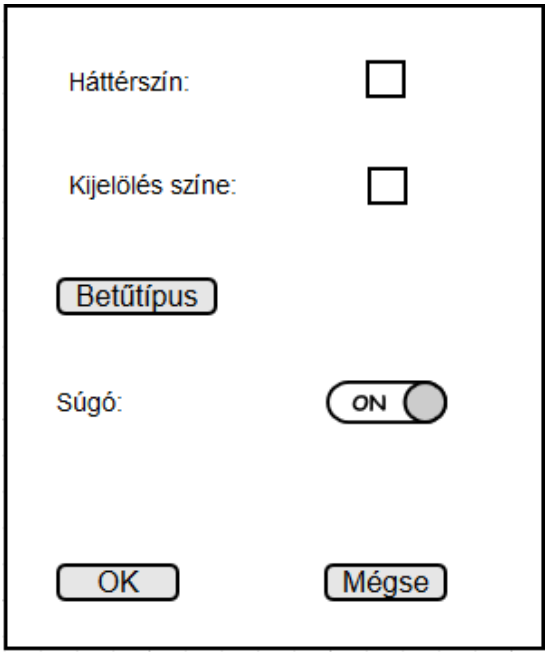
Ábra 17

A menü terve:



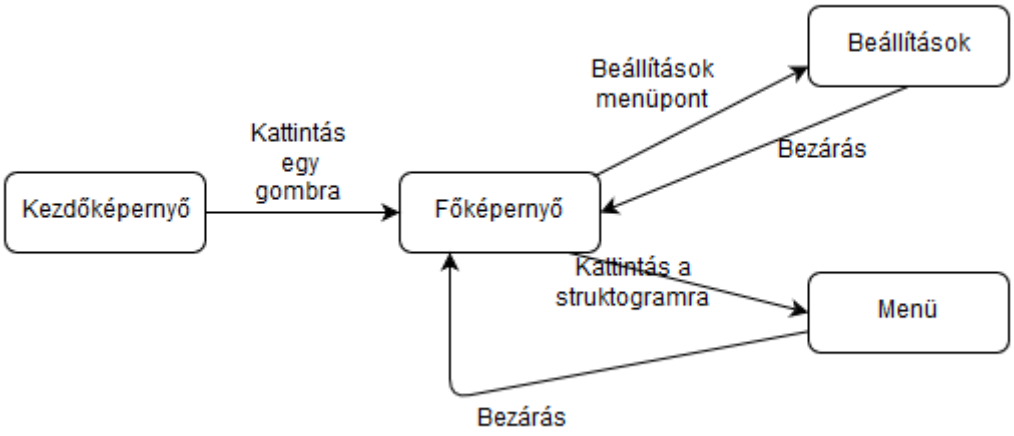
Ábra 18

A beállítások menü terve:



Ábra 19

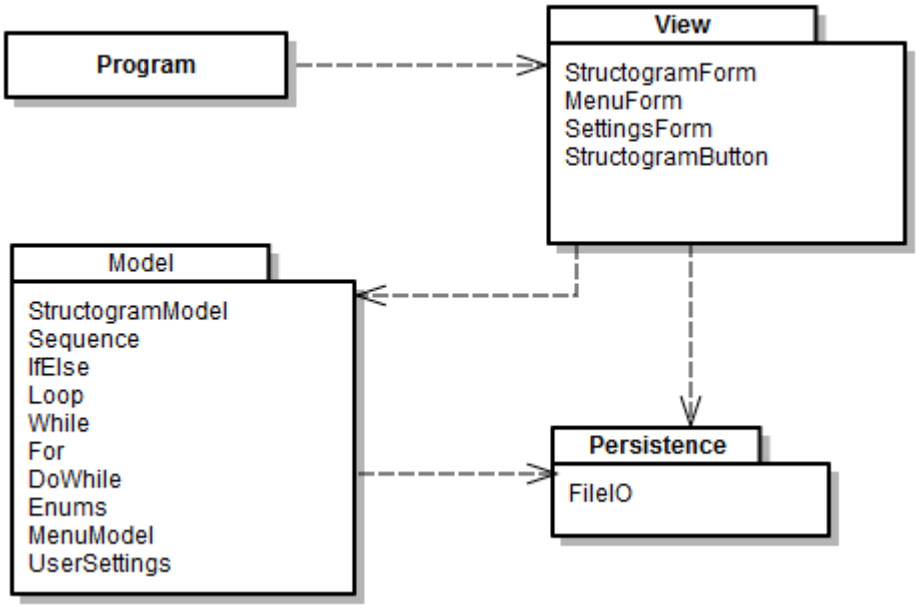
2.2.2. Képernyők közti navigálás



Ábra 20

A felhasználó a kezdőképernyőn valamelyik gombra kattintva jut el a főképernyőre. Itt rajzolódik ki a struktogram, ezen a felületen tud új struktogramot készíteni, a jelenlegit elmenteni, korábbit betölteni, kódot generálni belőle. Bármelyik struktogramelemre kattintva (kivéve a fejlécet), megnyílik a menü, ezen keresztül lehet új elemet hozzáadni, korábbit módosítani, törölni. A beállítások menüpontra kattintva megnyílik a beállítások ablak, ahol a struktogram kinézetét beállíthatja, illetve ki- és bekapcsolhatja a súgót.

2.3. A program szerkezete



Ábra 21

Az alkalmazás egy háromrétegű architektúrában lesz megvalósítva. A három réteg a modell, a nézet és a perzisztencia. Ennek megfelelően a program három package-re van osztva: Model, View, Persistence. Az adott rétegbe tartozó osztályok a nekik megfelelő package-ben találhatóak meg.

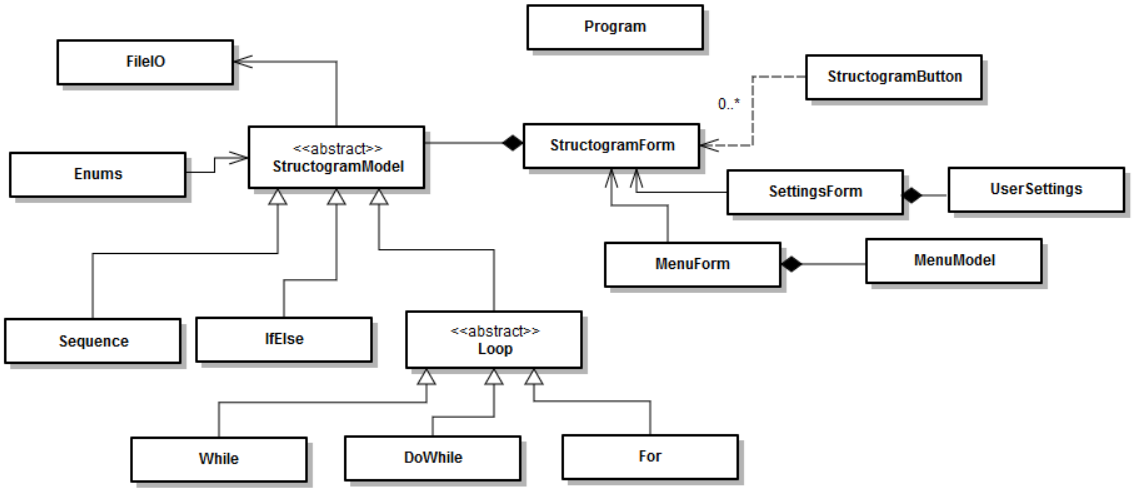
A modell réteg felelős a struktogram elemeinek reprezentálásáért, jellemzőik tárolásáért. A struktogramot egy fa szerkezetben tároljuk, a fa bejárási algoritmusai is itt találhatóak meg.

A nézet réteg egy grafikus felület, ez végzi a kommunikációt a felhasználóval. Ez a réteg jeleníti meg a struktogramot, a felhasználó ezen keresztül tud módosításokat végezni rajta.

A perzisztencia réteg felelős az adatok mentéséért és beolvasásáért. Ez végzi a struktogramok mentését, exportálását jpg-be, a struktogramok beolvasását, a beállítások mentését és az elkészített kód fájlba írását.

Részletesebb leírásuk a következő alfejezetben található meg.

2.3.1. Osztálydiagram



Ábra 22

2.4. Modell réteg

A struktogramokat felépítő elemeket egy fa szerkezetben fogjuk eltárolni. Minden csúcs tudja magáról, hogy ki a szülője, ki a gyereke (vagy gyerekei), és milyen szöveget tartalmaz a csúcs. Szekvencia esetén a csúcsnak csak egy gyereke van, az utána következő elem. Ciklus esetén (elöltesztelős, hátultesztelős és számlálós esetben is) két gyereke van, az egyik az az utasítás, amelyik a ciklus után jön majd, a másik pedig a ciklusmagnak az első eleme. Ehhez hasonlóan az elágazásnak három gyereke van: az elágazás után következő utasítás, az igaz részfa első eleme és a hamis részfa első eleme.
Ennek a fának a csúcsai a StructogramModel osztályból származtatott osztályok példányai.

2.4.1. StructogramModel osztály

A struktogram különböző típusú elemeinek reprezentálására a StructogramModel nevű absztrakt osztály és a belőle származtatott osztályok szolgálnak. A StructogramModel tárolja egy elem szülőjének és gyerekének a memóriacímét (parent, child), a saját azonosítóját (id), a megjelenítendő szöveget (label), valamint a csúcsból generált kódot (command). Az utóbbi kódgeneráláskor kap értéket. Mindegyik osztály, amelyik belőle származik, rendelkezik ezekkel az adattagokkal. Legfontosabb függvényei a TreeDepth(), az InsertNode(), a SearchIdInsertNode(), a DeleteNode(), a CreateCode(), és a FormatCode() függvények.

TreeDepth(): megadja, hogy mekkora annak a részfának a mélysége, amely csúcsot paraméterként átadunk neki.

InsertNode(): új csúcs beszúrásakor, illetve már létező csúcs lecserélésekor használjuk. Attól függően, hogy milyen típusú az új csúcs, létrehozza a hozzá tartozó objektumokat, majd meghívja a SearchIdInsertNode() függvényt, mely beszúrja az új csúcsot (csúcsokat) a megfelelő helyre. Szekvencia esetén csak egy elemet hozunk létre, de ciklus esetén kettőt, a ciklus feltételét és egy üres elemet a ciklusmagba. Hasonlóképpen elágazásnál létrehozzuk a feltétel elemet, valamint az igaz és a hamis ágba is egy-egy üres szekvenciát. Amennyiben nem új csúcsot akartunk hozzáadni a fához, hanem egy korábbit cseréltünk le, akkor a lecserélt elem és az új elem típusától függ, hogy mi fog történni. Ha mindkettő ciklus volt, akkor az új csúcs megőrzi az előzőnek a ciklusmagját és nem lesz új üres szekvencia benne. Ha pedig mindkettő elágazás volt, akkor az új megőrzi az előző igaz és hamis ágát is. Szekvencia esetén ilyennel nem kell foglalkozni.

DeleteNode(): kitörli a fából a paraméterül kapott azonosítójú csúcsot. Ha ciklust, vagy pedig elágazást törlünk ki, akkor a teljes részfájuk (a ciklusmag, illetve az elágazás igaz/hamis ága) is kitörlődik.

FormatCode(): egy absztrakt függvény, mivel ezt mindegyik típusnak másképpen kell implementálnia. Ez végzi el a fa egy adott csúcsának megfelelő C++ kód előállítását.

CreateCode(): a paraméterül megkapott csúcs részfájának meghívja a FormatCode()-ot, majd ezekből összerakja a teljes fának megfelelő kódot. Rekurzív módon járja be a fát, de azoknál a csúcsoknál, amiknek több gyereke is lehet (ciklus, elágazás), ezeknek a belső részfáit az adott osztály fogja bejárni, a FormatCode() függvény implementációjának megfelelően.

A TreeDepth(), a SearchIdInsertNode(), és a DeleteNode() függvények is rekurzív módon járják be a fát. Az első három függvény működése a következő: szekvencia esetén a rekurzió következő csúcsa a szekvencia gyereke lesz. Ezzel szemben ciklus és elágazás esetén először bejárjuk a részfájukat (ciklusmag, vagy pedig igaz ág, aztán hamis ág) és csak ezután jön a gyerekük, az a csúcs, amelyik utánuk jön.

2.4.2. Sequence osztály

A Sequence osztály a StructogramModel osztályból származik, ez felel meg a struktogramban a szekvencia elemeknek. Az ősosztályától örökölt adattagokon kívül tartalmaz még egy asszociatív tömböt, amelyre a kódgenerálás során van szükség. Ebben tárolja el az egyes változókat és a hozzájuk tartozó típust.

2.4.3. IfElse osztály

Az IfElse osztály is a StructogramModel osztályból származik, ez felel meg a struktogramban az elágazásoknak. Az örökölt adattagokon kívül még tartalmazza az igaz és a hamis részfájának az első elemének a memóriacímét (childTrue, childFalse).

2.4.4. Loop osztály

A Loop absztrakt osztály is a StructogramModel osztályból származik, ez felel meg a struktogramban a ciklusoknak. Az örökölt adattagokon kívül egy további adattagja van, a ciklusmag első elemének címe (child2). A ciklusokat azért érdemes egy osztályból származtatni, mert mindegyiknek ugyanazok az adattagjai, csupán a FormatCode() függvényük tér el, ami a ciklusból elkészíti a programkódot.

2.4.5. While, DoWhile és For osztályok

A While, a DoWhile és a For osztályok is a Loop osztályból származnak. Az elöltesztelős, a hátultesztelős, illetve a számlálós ciklus megfelelői a modellben.
Ezzel a különböző struktogramelemeknek megfelelő osztályok végére értünk.

2.4.6. Enums osztály

Az Enums osztály két felsoroló (enum) típusú adattagot tartalmaz (Types, Positions). A Types tartalmazza a különböző elemtípusokat, amik előfordulhatnak egy struktogramban (szekvencia, elágazás, elöltesztelős ciklus, hátultesztelős ciklus, számlálós ciklus). Arra szolgál, hogy az alkalmazásban könnyebben lehessen hivatkozni az egyes elemtípusokra. A Positions tartalmazza azokat a pozíciókat, ahova egy struktogram elemet be lehet szúrni (elé, helyette, utána). Erre is a kód jobb átláthatósága és karbantarthatósága miatt volt szükség.

2.4.7. MenuModel osztály

A MenuModel osztály ahhoz a grafikus felülethez tárolja az adatokat, amelyik akkor nyílik meg, ha rákattintunk egy elemre a struktogramban. Tárolja, hogy melyik pozícióba akarjuk beszúrni az új elemet (position), mi az alapértelmezett pozíció, ami a grafikus felületen alapértelmezetten be lesz jelölve (defaultPosition), milyen típusú

elemre kattintottunk (type), mi a beírt utasítás/feltétel (text), és azt, hogy elágazások esetén mi az igaz és a hamis ág szélességének aránya (trueFalseRatio). Ez utóbbi azt adja meg, hogy mekkora az igaz ág szélessége a teljes elemhez képest.

A MenuModel osztályra azért van szükség, mivel a grafikus felületből nem tudunk adatot kinyerni miután bezártuk, helyette ennek az osztálynak egy példányában elmentjük az adatokat és legközelebb, amikor újra rákattintunk egy elemre és megnyílik a menü, ebből az objektumból fogjuk a szükséges információkat megkapni (pl.: alapértelmezett pozíció).

2.4.8. UserSettings osztály

A UserSettings osztály a felhasználó által beállított beállítások tárolására szolgál. Ilyen a betűtípus, a háttérszín, a kijelölés színe, és hogy kér-e súgót. Ez az osztály tárolja a programban megjelenítendő súgó szövegeket is. Az alkalmazás bezárásakor elmentődik egy config.xml nevű fájlba. Indításkor, ha létezik ilyen nevű fájl az alkalmazás könyvtárában, akkor beolvassa. Ha nem létezik, akkor az alapértelmezett beállítások lesznek érvényesek (pl.: fehér háttérszín, 12-es betűméret, stb.).

2.5. Nézet réteg

A nézet réteg grafikus elemeit a Windows Forms keretrendszer objektumaival reprezentáljuk.

2.5.1. StructogramButton osztály

A StructogramButton osztály példányai felelnek meg a struktogramot alkotó téglalapoknak. Tárolja, hogy az alakzat melyik oldalán milyen vastagságú szegély legyen (top, left, bottom, right), milyen a gomb típusa (type), a háttérszínt (drawColor) és az igaz/hamis ágak arányát (trueFalseRatio). Az utóbbira csak elágazás esetén van szükség. Szekvencia esetén egy egyszerű téglalapot rajzolunk ki. Elágazás esetén a két sarokba az igaz és hamis ágat jelző kis ferde vonalakat is berajzoljuk. A ciklusok két téglalapból tevődnek össze, a feltételt tároló téglalap és a ciklus „szára”, a függőleges oszlop. A fejléc esetén nem téglalapot, hanem egy oválist rajzol ki, amiből lefele kinyúlik egy vonal, ez köti össze a struktogram többi részével.

Az objektum kirajzolódásakor meghívódik az OnPaint() metódus, ami a gomb típusától függően egy téglalapot (DrawSequence()), egy elágazást (DrawIfElse()), vagy

pedig egy fejlécet (DrawHeader()) fog kirajzolni. A megjelenített alakzatokba a szöveg középre igazítását és kiírását a WriteText() metódus végzi.

2.5.2. StructogramForm osztály

A StructogramForm jelenik meg az alkalmazás elindításakor, ez végzi a struktogram kirajzolását. Indításkor paraméterként megkap egy StructogramModel objektumot, ezt eltárolja a tree adattagjában, egy MenuModel objektumot és egy UserSettings objektumot, ezeket pedig a menuModel és a settings adattagjaiban tárolja. A felső sorba elhelyezünk egy menüt, az ablak többi része a struktogram megjelenítésére szolgál. A struktogram kirajzolását többek között a PaintTree() metódus végzi. Végigmegy a fán és az egyes csúcsaira meghívja a típusának megfelelő kirajzoló függvényt (AddSequence(), AddIfElse(), AddLoop()). Azon típusoknak, akiknek több gyereke is van (elágazás, ciklus), a többi gyerekének a részfáján nem megy végig, azt majd a kirajzoló függvényük végzi el, hanem folytatja az utána következő elemmel. A fejléc kirajzolását a DrawHeader() metódus végzi, amennyiben van fejléce az algortimusnak.

AddSequence(): kirajzol egy szekvencia elemet.

AddIfElse(): először kirajzolja a feltételt tartalmazó téglalapot, majd bejárja először az igaz ágát, azt kirajzolja, majd a hamis ágat, és azt is megjeleníti. A két ágban a gombok magassága úgy van megállapítva, hogy ha az egyik oldalon kevesebb elem lenne, akkor ott annyival legyen nagyobb a gombok magassága, hogy összességében ugyanolyan magas legyen a két ág.

AddLoop(): működése attól függ, hogy elöltesztelős, vagy pedig hátultesztelős ciklust rajzolunk ki. Az elöltesztelős és a számlálós között nincs semmilyen megkülönböztetés kirajzoláskor. Mindhárom esetben nagyon hasonlóképpen működik a függvény, csak annyi a különbség, hogy hátultesztelős ciklusnál a ciklusfeltételt tartalmazó téglalap alulra kerül, nem pedig felülre, és a ciklus szélén lévő oszlop feljebb kerül, míg elöltesztelősnél lejjebb van. Mindegyik esetben bejárja a ciklusmag által alkotott részfát, majd pedig kirajzolja, ennek pozíciója független a típustól.

Ha rátoljuk az egeret valamelyik téglalapra, és a súgó be van kapcsolva, akkor megjelenik egy buborék, benne a súgó szövegével. Ezt a CreateHint() függvény hozza létre. A gomb típusától függően változik a súgó szövege. Fejlécnél és számlálós ciklusnál megmutatja, hogy milyen szintaktikával érdemes a tartalmukat beírni az optimális kódgeneráláshoz. Az elem típusától függetlenül segítséget nyújt a felhasználónak a program

használatához (pl.: menü megnyitásához mit kell tenni, fejlécet hogyan lehet hozzáadni, stb.).

2.5.3. MenuForm osztály

A MenuForm osztály valósítja meg azt a menüt, amely egy struktogram elemre kattintva megjelenik. Eltárolja egy adattagba (model) a MenuModel egy példányát, a gombot, amire kattintottunk (button), hogy történt-e módosítás (modified), illetve, hogy új csúcsot kell-e beszúrni (inserted). A menüben van három checkbox, ezekkel lehet kiválasztani az új elem pozícióját, egy legördülő menü, ezzel az új elem típusát lehet megadni, és egy text box, ahova a feltételt/utasítást lehet beírni. Az OK gomb hatására a megadott adatoknak megfelelő új elem létrejön és beszúródik a helyére. A Mégse gomb megnyomásakor visszatérünk a struktogramot megjelenítő felületre, de nem történik semmi a struktogrammal. A Törlés gomb hatására kitörlődik a fából az az elem, amelyre kattintva megnyílt a menü. A beszúrást és a törlést is a StructogramModel osztály megfelelő metódusai végzik el.

2.5.4. SettingsForm osztály

A SettingsForm osztály a struktogram kinézetének beállítására való, továbbá a súgót is itt lehet ki- és bekapcsolni. Egy adattagjában eltárolja a UserSettings osztály egy példányát (settings), ebbe menti el az új beállításokat.

2.6. Perzisztencia réteg

A perzisztencia rétegben lévő FileIO osztály felelős az adatok fájlba írásáért és beolvasásáért.

WriteTreeToFile(): elvégzi a paraméterül kapott fa fájlba írását. A fájl szerkezete a következő: Az első sorba van beírva, hogy mekkora volt a legnagyobb ID a fában, erre majd a beolvasáskor lesz szükség. A második sor tartalmazza az elágazások igaz/hamis ágainak a szélesség arányait „WidthRatios: <ID>,<Arány>;<ID2>,<Arány2>;..” alakban. Ezután a fa minden egyes csúcsára meghívja a WriteNode() metódust, ami az egyes csúcsokat fájlba írja. A sorokba pedig a fa egyes csúcsainak a tulajdonságait írja a következő alakban:

Szekvencia esetén: <ID>;<Típus>;<Label>;<ChildID>

Elágazás esetén: <ID>;<Típus>;<Label>;<ChildID>;<TrueChildID>;<FalseChildID>

Ciklus esetén: <ID>;<Típus>;<Label>;<ChildID>;<Child2ID>
Ha nincs gyereke egy csúcsnak, akkor az ő azonosítójának a helyére -1 kerül.

ReadTreeFromFile(): a paraméterül kapott fájlt beolvassa és létrehozza a neki megfelelő fát. Ha nem megfelelő a fájl szerkezete, akkor szól a felhasználónak és visszatér a program abba az állapotba, mielőtt megpróbált volna betölteni egy struktogramot. Ha megfelelő a fájl szerkezete, akkor eltárolja a legnagyobb ID-t (fájl első sora), az elágazások szélességarányait (második sor), majd pedig a fák csúcsait leíró sorok alapján létrehozza a nekik megfelelő objektumokat és beállítja köztük a szülő-gyerek kapcsolatokat. Egy adott azonosítójú csúcs gyerekeit a CreateChild() függvény hozza létre.

WriteCode(): a paraméterül kapott fának megfelelő programkódot írja a paraméterül kapott fájlba. A kódgenerálás egy másik fejezetben lesz részletesebben kifejtve.

SaveSettings(): egy config.xml fájlba elmenti a paraméterül kapott UserSettings objektumot, a LoadSettings() függvény pedig ugyanebből a fájlból betölti az adatokat.

2.7. Kódgenerálás

A kódgenerálást a StructogramModel osztályból származtatott osztályok FormatCode() függvényei végzik. Minden osztályban másképp van implementálva, az osztály jellemzőitől függően.

2.7.1. Típusok

A felhasználó megmondhatja a változóiról, hogy milyen típusúak legyenek. Ilyen esetben a program nem ellenőrzi, hogy az a típus létezik-e C++-ban. Így a felhasználó olyan típusokat is használhat, amiket csak a kód legenerálása után fog létrehozni a már kész kódban.

Nem deklarált változók típusát az olyan első alkalommal határozza meg a program, amikor olyan értéket kapnak, amiből megállapítható a típus. A későbbi értékadásoknál nem ellenőrzi, hogy megfelelő típusú értéket akar-e benne eltárolni, ez a felhasználó felelőssége. A program a következő típusokat tudja felismerni: int, float, string, char, bool.

Int típusú akkor lesz a változó, ha az értékadás jobb oldalán egy egész szám áll, vagy pedig egész típusú változó.

Float típusú akkor lesz a változó, ha az értékadás jobb oldalán egy tört szám áll (pl.: 1.2, 0.3, stb., azaz legalább egy számjegyet követően van benne egy pont, majd ezután is legalább egy számjegy következik), vagy pedig float típusú változó.

String típusú akkor lesz a változó, ha az értékadás jobb oldalán idézőjelek között van bármilyen karaktersorozat, vagy pedig string típusú változó áll a jobb oldalon.

Char (azaz karakter) típusú akkor lesz a változó, ha az értékadás jobb oldalán egy karakter áll aposztrofok között, vagy pedig char típusú változó.

Bool (azaz logikai) típusú akkor lesz a változó, ha az értékadás jobb oldalán a következő szavak valamelyik áll: igaz, hamis, true, false. Vagy pedig akkor, ha a jobb oldalon bool típusú változó áll.

Ha egy értékadás jobb oldalán egy olyan tömbnek áll egy eleme, aminek tudjuk a típusát, akkor a baloldalon álló változó a tömbelem típusát fogja megkapni.

2.7.2. Szekvencia

A szekvenciákból történő kód elkészítését a Sequence osztály végzi.
Ha az utasítás úgy kezdődik, hogy típus és változó, akkor az adott változót eltároljuk a megadott típussal.

Ha a változó típusának megadása nélkül értéket kap, akkor, amennyiben lehetséges, az egyenlőségjel jobb oldala alapján megállapítjuk a változó típusát, majd eltároljuk, hogy a későbbiekben használhassuk és hozzáfűzzük az utasítás elejére a típust.

Minden esetben megszüntetjük a fölösleges szóközöket, a legyen egyenlőt (:=) lecseréljük egyenlőre, a szükséges mértékben megnöveljük előtte az indentációt és a végére hozzáfűzünk egy pontosvesszőt és egy sortörést.

2.7.3. Elágazás

Az IfElse osztály által generált kód a következőképpen áll össze.

Először elkészítjük a feltételt. A fa adott csúcsának tartalmát formázzuk úgy, hogy a struktogramban használt jelöléseket lecseréljük a C++ nyelvben használtakra. Egy darab egyenlőség helyett kettőt írunk, az „és”, illetve „and” szavakat lecseréljük „&&”-ra, hasonlóképpen a „vagy” és „or” szavakat pedig lecseréljük „||”-ra. A feltétel elejére kellő indentáció beszúrása után következik az „if” kulcsszó, majd zárójelek közé beírjuk az előbb megformázott feltételt.

Ezután a kapcsos zárójelek közé kerül az elágazás igaz ága. Bejárjuk az elágazás igaz ágának részfáját és összeállítjuk belőle a kódot. Mindegyik csúcs a saját típusának megfelelő kódelőállító függvényt fogja meghívni, így a végeredményben megkapjuk a kódot, amit csak be kell illeszteni a kapcsos zárójelek közé.

Teljesen ugyanígy állítjuk elő a hamis ág kódját is. Az „else” kulcsszó utáni kapcsos zárójelek közé illesztjük majd be. Ekkor az elágazás hamis részfáját járjuk be.

2.7.4. Ciklus

Elöl- és hátultesztelős ciklus esetén is a ciklusfeltételt ugyanúgy formázzuk, mint az elágazás esetében (ezt a formázást a StructogramModel osztály FormatLabel() függvénye végzi el).

Elöltesztelős ciklus esetén a „while” kulcsszó után zárójelek közé beírjuk a formázott ciklusfeltételt, majd a kapcsos zárójelek közé illesztjük be a ciklusmag kódját. Az utóbbit úgy állítjuk elő, hogy bejárjuk a ciklusmagot jelentő részfát, mindegyik csúcsára meghívva a megfelelő kódgeneráló függvényt.

A hátultesztelős ciklus csak annyiban tér el az előzőtől, hogy itt a „do” kulcsszó kapcsos zárójelek közé írjuk a ciklusmag kódját, majd következik a „while” kulcsszó, ezután pedig zárójelek között a ciklusfeltétel.

Számlálós ciklus esetén a ciklusfeltételt felbontjuk a ciklusváltozóra, a kezdőindexre, az utolsó indexre és a léptékre. A ciklusváltozó mindig int típusú lesz, ezért a felhasználónak ezt nem kell megadnia. Ha nincs lépték megadva, akkor +1 lesz az értéke. Az alkalmazás nem ellenőrzi, hogy a felhasználó értelmes intervallumot adott-e meg (pl.: [2;10] intervallumon -2-es lépték esetén értelmetlen, de nem fog szólni a program), viszont szól, ha a ciklusfeltétel formája nem felel meg az elvárásnak (ciklusváltozó = kezdőIndex..utolsóIndex (lépték)). A kód úgy áll elő, hogy a „for” kulcsszót követően zárójelek közé beírjuk a megfelelő elemekre szétbontott és azokból összeállított ciklusfeltételt, majd kapcsos zárójelek közé beírjuk a ciklusmag kódját ugyanúgy, mint a többi ciklus típus esetén. Ha a ciklusfeltétel formája nem az elvárt volt, akkor minden formázás nélkül egy az egyben beírjuk a feltétel helyére a kódban.

2.7.5. Kód összeállítása

A programkód legenerálásakor a struktogramot reprezentáló fa gyökeréből indulunk el. A fát bejárva minden csúcsának elkészítjük a neki megfelelő kódot.

Ha van fejléce az algoritmusnak, akkor a fejléc tartalmát odafűzzük a kód elejére és az azt követő kapcsos zárójelek közé illesztjük be a korábban összeállított kódot.

Ha nincsen fejléce, akkor is hasonlóan járunk el, csak a fejléc tartalma helyett az „int main()” szöveg kerül majd a kód elejére, és ezután, az előbbihez hasonlóan kapcsos zárójelek közé beillesztjük a struktogram alapján elkészített kódot.

Ezek után még két módosítást végzünk a kódon. Egyrészt az összes ékezetes betűt kicseréljük a neki megfelelő ékezet nélkülire, másrészt a legelejére kommentként odakerül a dátum, amikor a kód fájlba íródik.

3. Tesztelés

A teszteseteket két részre bontjuk, modultesztekre és rendszertesztekre.

3.1. Modultesztek

A modell réteg automatikus egység tesztjei a StructogramModelTests osztályban találhatóak, a grafikus felület egységtesztelésére nem készültek automatikus tesztesetek, ezeket kézzel végeztük el.

3.1.1. Automatikus tesztesetek

- StructogramModelTest(): létrehoz egy struktogram elemet, és ellenőrzi, hogy a megadott adatok valóban eltárolódnak-e
- AddChildTest(): ellenőrzi, hogy egy elemnek megadva egy gyereket, valóban eltárolódik-e a szülőnél a gyerek címe és a gyereknél a szülőé
- FormatLabelTest(): egy szövegben a különböző logikai műveletek (és, vagy, egyenlő-e) C++-ban használtakra kicserélését ellenőrzi
- AddHeaderTest(): létrehoz egy struktogram elemet, majd hozzáad egy fejlécet. Ezek után ellenőrzi, hogy a fejléc rendelkezik-e az elvárt tulajdonságokkal
- RemoveHeaderTest(): kitörli egy struktogram fejlécét és megnézi, hogy ekkor a fa gyökere visszaáll-e az eredeti állapotba
- SequenceFormatCodeTest(), SequenceFormatCodeTest2(): a Sequence osztály által generált kódot ellenőrzik, különös hangsúllyal a változók típusmeghatározására
- HeaderCodeTest(): a struktogram fejlécéből készített kódot ellenőrzi egy mintán
- IfElseFormatCodeTest(): az elágazásokból készített kód ellenőrzése egy mintán
- WhileFormatCodeTest(): elöltesztelős ciklusból generált kód ellenőrzése egy mintán
- DoWhileFormatCodeTest(): hátultesztelős ciklusból generált kód ellenőrzése egy mintán
- ForFormatCodeTest():hátultesztelős ciklusból generált kód ellenőrzése a különböző előfordulható szintaktikák szerint

3.1.2. Grafikus felület egységtesztjei

3.1.2.1. Új szekvencia hozzáadása előző után

- Esemény: struktogram egyik elemére kattintás, menüben a következő beállítások: pozíció: utána, típus: Szekvencia, utasítás: i:=1
- Elvárt eredmény: jelenjen meg az után az elem után, amire kattintottunk egy szekvencia i:=1 szöveggel
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.2. Új szekvencia hozzáadása előző elé

- Esemény: struktogram egyik elemére kattintás, menüben a következő beállítások: pozíció: elé, típus: Szekvencia, utasítás: i:=1
- Elvárt eredmény: jelenjen meg az előtt az elem előtt, amire kattintottunk egy szekvencia i:=1 szöveggel
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.3. Új szekvencia hozzáadása előző helyett

- Esemény: struktogram egyik elemére kattintás, menüben a következő beállítások: pozíció: helyette, típus: Szekvencia, utasítás: i:=1
- Elvárt eredmény: jelenjen meg a helyett az elem helyett, amire kattintottunk egy szekvencia i:=1 szöveggel
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.4. Új elágazás hozzáadása előző elem elé

- Esemény: struktogram egyik elemére kattintás, menüben a következő beállítások: pozíció: helyette, típus: Elágazás, utasítás: i<3, szélesség: 50 %
- Elvárt eredmény: jelenjen meg az elé az elem elé, amire kattintottunk egy elágazás i<3 szöveggel, egyenlő széles igaz-hamis ágakkal
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.5. Új elágazás hozzáadása előző elem helyett

3.1.2.6. Új elágazás hozzáadása előző elem után

3.1.2.7. Új elem hozzáadása az elágazás igaz ágába

3.1.2.8. Új elem hozzáadása az elágazás hamis ágába

3.1.2.9. Új elöltesztelős ciklus hozzáadása előző elem elé

3.1.2.10. Új elöltesztelős ciklus hozzáadása előző elem helyett

3.1.2.11. Új elöltesztelős ciklus hozzáadása előző elem után

3.1.2.12. Új szekvencia hozzáadása az elöltesztelős ciklus magjába

3.1.2.13. Új hátultesztelős ciklus hozzáadása előző elem elé

3.1.2.14. Új hátultesztelős ciklus hozzáadása előző elem helyett

3.1.2.15. Új hátultesztelős ciklus hozzáadása előző elem után

3.1.2.16. Új szekvencia hozzáadása az hátultesztelős ciklus magjába

3.1.2.17. Új számlálós ciklus hozzáadása előző elem elé

3.1.2.18. Új számlálós ciklus hozzáadása előző elem helyett

3.1.2.19. Új számlálós ciklus hozzáadása előző elem után

3.1.2.20. Új szekvencia hozzáadása az számlálós ciklus magjába

3.1.2.21. Új szekvencia hozzáadása, utasítás kihagyása

- Esemény: struktogram egyik elemére kattintás, menüben a következő beállítások: pozíció: utána, típus: Szekvencia, utasítás: üresen hagyjuk
- Elvárt eredmény: ne engedje a program így létrehozni az elemet
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.22. Fejléc hozzáadása

- Esemény: jobb gombbal kattintva kiválasztjuk a Fejléc hozzáadása menüpontot
- Elvárt eredmény: jelenjen meg egy fejléc (egy ovális alakzat egy rövid vonallal alul) a struktogram fölött
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.23. Fejléc törlése

- Esemény: jobb gombbal kattintva kiválasztjuk a Fejléc törlése menüpontot
- Elvárt eredmény: ne jelenjen meg fejléc a struktogram fölött
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.24. Struktogram mentése

- Esemény: Struktogram mentése menüpontra kattintás
- Elvárt eredmény: jelenjen meg egy fájlválasztó ablak, fájl kiválasztása után mentse el a struktogramot
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.25. Struktogram betöltése helytelen bemenettel

- Esemény: Struktogram betöltése menüpontra kattintás, nem megfelelő szerkezetű fájl kiválasztása
- Elvárt eredmény: szóljon a program, hogy nem megfelelő a fájl szerkezete, térjünk vissza abba az állapotba, ami előtt be akartuk tölteni a fájlt
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.26. Kódgenerálás

- Esemény: Kódgenerálás menüpont kiválasztása
- Elvárt eredmény: jelenjen meg egy fájlválasztó ablak, fájlnév megadása után írja bele a struktogramból generált kódot
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.1.2.27. Beállítások menü

- Esemény: Beállítások menüpont kiválasztása, ott következő adatok megadása: háttérszín: piros, kijelölés színe: sárga, betűtípus: 16-os méretű kék, súgó kikapcsolása
- Elvárt eredmény: a struktogram háttérszíne legyen piros, a kijelölt elem színe sárga, a betűk 16-os méretűek és kékek, valamint ne jelenjen meg súgó, ha ráhúzzuk a kurzort egy elemre
- Kapott eredmény: megegyezik az elvárt eredménnyel

3.2. Rendszerteszt

A rendszerteszt során a teljes rendszer együttes működését vizsgáljuk. Ehhez válasszuk a beszúró rendezés algoritmusát, melynek megépítése során egy viszonylag összetett struktogramot készítünk, majd kódot generálunk belőle, minél több beállítást és menüpontot kipróbálva közben.

A struktogram elkészülte után adjuk meg a fejlécét is, határozzuk meg a Beállítások menüben a struktogram kinézetét, majd exportáljuk jpg-ként. Ezután mentsük el a struktogramot és zárjuk be az alkalmazást.

Újra elindítva az alkalmazást, töltsük be az előbbi struktogramunkat (ezáltal ellenőrizve a mentés és a betöltés funkciókat). Ezek után generáljunk belőle kódot, majd nézzük meg, hogy a generált kód megfelel-e az elvárásoknak (típusokat helyesen állapítja-e meg, ahol lehet, indentálás megfelelő-e, stb.).

4. Továbbfejlesztési lehetőségek

4.1. Felület

- Több struktogramot is meg tudjon jeleníteni egy ablakban
- Felhasználók létre tudjanak hozni saját típusokat
- A struktogram minden eleme egyesével kézzel állítható szélességű és magasságú legyen

4.2. Kódgenerálás

- Műveletek és egyéb kifejezések típusát tudja megállapítani
- Felhasználó által definiált típusokból tudjon kódot készíteni
- Több nyelvre is lehessen kódot generálni

5. Irodalomjegyzék

1. C# dokumentáció: https://msdn.microsoft.com/en-us/library/kx37x362.aspx
2. Wikipédia: https://en.wikipedia.org/wiki/Tree_%28data_structure%29
3. http://tamop412.elte.hu/tananyagok/algoritmusok/index.html