

SmartPaint

Okos rajzprogram - Szoftverarchitektúrák házi feladat

Készítették:

Hegedűs Tamás László

Dusza Andrea

Tartalom

[1 Specifikáció 2](#_Toc405162629)

[1.1 Feladatkiírás 2](#_Toc405162630)

[1.2 Részletes leírás 2](#_Toc405162631)

[1.3 Technikai paraméterek 3](#_Toc405162632)

[1.4 Szótár 4](#_Toc405162633)

[1.5 Use-case diagram 5](#_Toc405162634)

[2 Tervezés 6](#_Toc405162635)

[2.1 Architektúra 6](#_Toc405162636)

[2.2 UI terv 7](#_Toc405162637)

[3 Megvalósítás 9](#_Toc405162638)

[3.1 Osztálydiagram 9](#_Toc405162639)

[3.2 Képernyőképek 9](#_Toc405162640)

[3.2.1 Alapvető műveletek alakzatokkal 9](#_Toc405162641)

[3.2.2 Ecset és radír 12](#_Toc405162642)

[3.2.3 Transzformációk 15](#_Toc405162643)

[4 Összefoglalás 16](#_Toc405162644)

[4.1 Kihívások, tanulságok 16](#_Toc405162645)

[4.2 Továbbfejlesztési lehetőségek 16](#_Toc405162646)

[5 Telepítési útmutató 16](#_Toc405162647)

[6 Hivatkozások 17](#_Toc405162648)

# Specifikáció

## Feladatkiírás

A feladat célja egy olyan rajzprogram létrehozása, mely pixel halmazok formájában adott foltok szerkesztésére, valamint ezeket manipuláló algoritmusok futtatására alkalmas. A rajzprogramban ezeket a foltokat lehet átrajzolni (pixelek törlése vagy hozzáadása), valamint ezeken a foltokon lehet egyszerű műveleteket lefuttatni (például kettő összeolvasztása, nem teljesen körbevett pixelek törlése). (A cél nem bonyolult képmorfológiai műveletek megvalósítása, hanem az ezek befogadására képes, kényelmesen bővíthető (transzformáció plug-in-eket támogató) architektúra kialakítása.)

A program alapvető funkciói:

- Kép betöltése

- Egy "kép -> ponthalmazok halmaza" művelet lefuttatása (a foltok megkeresése)

- Egy "folthalmaz -> folthalmaz" művelet paraméterezése és lefuttatása

- Folthalmaz elmentése, visszatöltése

- Foltok kijelölése

- Foltokból pixelek törlése (radír) vagy hozzáadása (ecset)

Preferált platform: Qt C++ vagy WPF.

## Részletes leírás

A rajzprogram .png, .jpg, és .bmp formátumú nyers képek, illetve az általunk definiált .spt kiterjesztésű SmartPaint projektfájlok (folthalmaz-fájlok) kezelésére lesz képes. Az alkalmazás indításkor azonnal azt kéri, hogy hozzunk létre egy új projektet, vagy a fájlrendszerből töltsünk be egy meglévőt. Ezután a projekten az alábbi műveleteket végezhetjük:

* Meglévő foltokon különböző transzformációk végzése
* Foltok importálása a projektbe egy másik, már létező projekt fájlból
* Nyers kép betöltése által új foltok létrehozása
* A projekt foltjainak exportálása .png formátumú képfájlba

Az alkalmazás alapvetően három folthalmaz-transzformációt tesz lehetővé: összefüggő alakzatokra bontást, összeolvasztást és eróziót, ezek menüpontok formájában elérhetőek a felhasználó számára. Mozgatás után az egyes foltok egymást átfedhetik, ezért a foltok között egy kirajzolási sorrend lesz definiálva. A program képes lesz bővítmények kezelésére olyan módon, hogy a fájlrendszerben a megfelelő mappába elhelyezett dll-eket indításkor detektálja és betölti, ezután az új dll által nyújtott plusz transzformációkat is elérhetővé teszi a felhasználói felületen.

Az összefüggő alakzatokra bontás úgy történik, hogy a program felderíti a foltokat a kép pixeleinek átlátszósága alapján: a teljesen átlátszó pixelek nem lesznek folt részei, a többi közül pedig egybefüggő halmazok alkotnak egy-egy foltot.

Ami az alkalmazás megvalósítását illeti: Model-View-ViewModel felépítésű lesz.

## Technikai paraméterek

A definiált alkalmazást .NET platformra, WPF felhasználásával C# nyelven készítjük el. Csak .NET keretrendszerrel rendelkező, WPF-et támogató - például Windows 7 és 8 operációs rendszeren - lesz futtatható. Az alkalmazásnak nem lesz különösebb hardverigénye. A fejlesztés és tesztelés az alábbi konfigurációjú számítógépen fog történni:

-Windows 8.1 x64 operációs rendszer

-Intel Core i5-2450M CPU @ 2.50 GHz

-6GB RAM

-ATI Mobility Radeon HD7650M 2GB videokártya

-1600X900 felbontású kijelző

## Szótár

**Pixel:** Színnel és koordinátákkal rendelkező képpont. Egy pixel pontosan egy foltnak a része.

**Szomszédos pixelek:** A pixel a közvetlenül fölötte, alatta, mellette lévő pixelekkel, vagyis összesen legfeljebb 4 másikkal lehet szomszédos.

**Nem teljesen körbevett pixel:** Olyan pixel, amelynek 4-nél kevesebb szomszédja van a foltban.

**Folt:** Egy vagy több pixelből álló, nem feltétlenül egybefüggő pixelhalmaz.

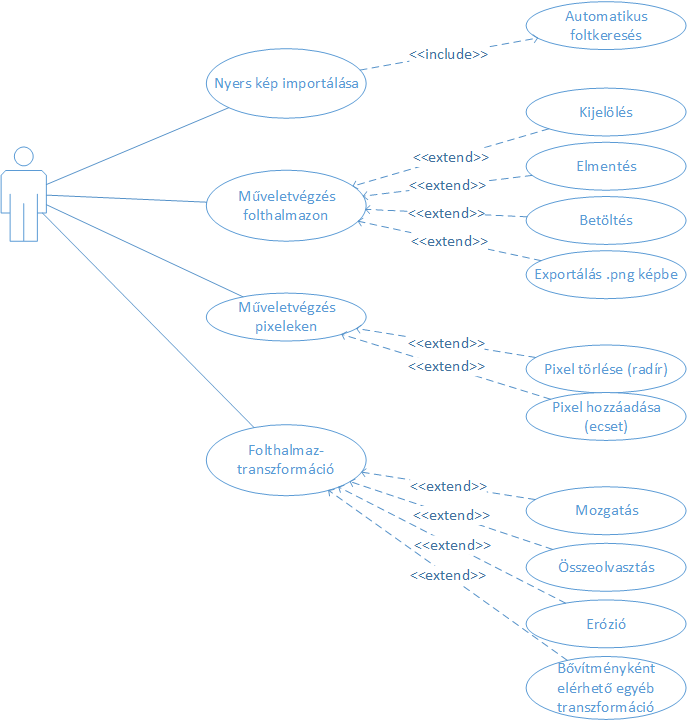
**Folthalmaz:** Több különböző folt együttese.

**Összeolvasztás**: Az éppen kijelölt folthalmazból egyetlen foltot előállító művelet, melynek elvégzése után az összes kiválasztott pixel egy foltot fog alkotni.

**Erózió**: Folthalmazon végezhető transzformáció, melynek során a nem teljesen körbevett pixelek eltávolításra kerülnek az egyes foltokból.

## Use-case diagram

A program az alábbi funkciókat támogatja:



1. ábra: Use-case diagram

# Tervezés

## Architektúra

Az alkalmazás háromrétegű, Model-View-ViewModel (röviden: MVVM) architektúrájú.



2. ábra: MVVM architektúra (Forrás: Wikipédia)

Az MVVM-et a Microsoft vezette be kimondottan a WPF és a Silverlight számára, az MVC minta elveit követi, de kihasználja az adatkötési modell (data binding model) teremtette laza csatolás lehetőségét.

A Model az MVC mintához hasonlóan adat-absztrakció. Adatok betöltése, mentése, törlése és módosítása a modellen keresztül hajtható végre. A Model az MVC mintával ellentétben csak aViewModellel van kapcsolatban, ezzel a View és Model közötti függőség megszünik.

A View alatt a felhasználói felület leírását értjük, mind a deklaratív XAML (Extensible Application Markup Language), mind a mögöttes kód fájlok (code-behind files) összességét. A mögöttes fájlok a felhasználói felülethez közvetlenül kötődő kódokat tartalmazzák.

A ViewModel feladata a legösszetettebb. A ViewModelnek kell a modellből az adatokat eljuttatni a nézet felé, illetve a nézet eseményeire - melyek a ViewModelben függvényhívások lesznek - a szükséges adatokat a modell felé propagálnia. Emellett a ViewModel végezhet adat validálást, sőt akár a View állapotát is tárolhatja, ami több szempontból is előnyös lehet. Összetett View esetén az állapotokat érdemes a ViewModelben tárolni, mert logikailag ehhez külön Model-t létrehozni csak ritkán szükséges.

Az MVVM minta erejét és újszerűségét az adatkötési motor (binding engine) adja, melynek révén a View és a ViewModel csak lazán kapcsolódik, azaz a fejlesztésük teljesen függetlenül is történhet. Így lehetővé válik, hogy a felhasználó felület- és a funkcionális egységek fejlesztése egy időben, felesleges függőségek nélkül, biztonságosan és könnyen tesztelhetően megvalósuljon.

A következő ábra már a SmartPaint alkalmazás konkrét szerkezetét mutatja be.

3. ábra: A SmartPaint arcitektúrája

ADAT RÉTEG

Perzisztencia

Konverterek

ÜZLETI RÉTEG

Modell

Bővítmények

PREZENTÁCIÓS RÉTEG

ViewModel

View – UI komponensek

Erózió

Foltkeresés

Összeolvasztás

…

Koordinálás

Logolás

Az adatréteg legfontosabb feladata, hogy a program által kezelt projektek mentési formátumát definiálja, és végrehajtja a konverziót mindkét irányba (mentés és betöltés). Jelenleg csak a fájlrendszerre való mentés lehetséges, adatbázisba, felhőbe, stb. viszont nem.

Az üzleti réteg felelős az aktuálisan szerkesztés alatt lévő dokumentum integritásáért.

Mivel egy rajzprogramról van szó, a prezentációs rétegre különösen nagy hangsúly helyeződik. A prezentációs réteg feladata, hogy az alakzatokat megjelenítse a felhasználó számára, és interfészt nyújtson a számára, amin keresztül az alakzatokat manipulálhatja.

A fejlesztés során felhasznált fontosabb tervezési minták:

* MVVM: ezt már részletesen bemutattam.
* Dependency inversion: Ahelyett, hogy az alkalmazás expliciten hivatkozna a bővítményekre, a bővítmények hivatkoznak az alkalmazás által definiált interfészre.
* Singleton: az ApplicationContext osztálynak van egy példánya, ami koordinálási feladatokat végez, összeköti a különböző rétegek objektumait. Az, hogy alkalmazásszinten csak egy példány létezik belőle, a Singleton minta használatát mutatja.
* Extensibility: a bővítmény dll-ek dinamikus betöltése ezen mintát is követi.

## UI terv

A felhasználói felülethez WPF-et (Windows Presentation Framework) használunk, ami a legtöbb felhasználó számára ismerősnek tűnő külső elkészítését teszi lehetővé, és számos előre elkészített komponenst támogat. A modularitás és a komponensek deklaratív leírása jelentősen megkönnyíti a fejlesztői csapat dolgát.

A felhasználói felület az aktuális Microsoft stílusnak megfelelően két menüsort fog tartalmazni, amik közül a felső szöveges menüpontokat tartalmaz, az alsó pedig grafikus ikonokat, amikkel a leggyakrabban használt funkciók hajthatóak végre.

A menük kialakításakor odafigyelünk az ergonómiából tanult 7+-2 szabályra, vagyis arra, hogy a felhasználók átlagosan 7 dolgot tudnak megjegyezni és elkülöníteni. Ez azt jelenti, hogy egyik menübe sem szerencsés 7-nél több menüpontot tenni, 9-nél többet pedig szinte tilos!

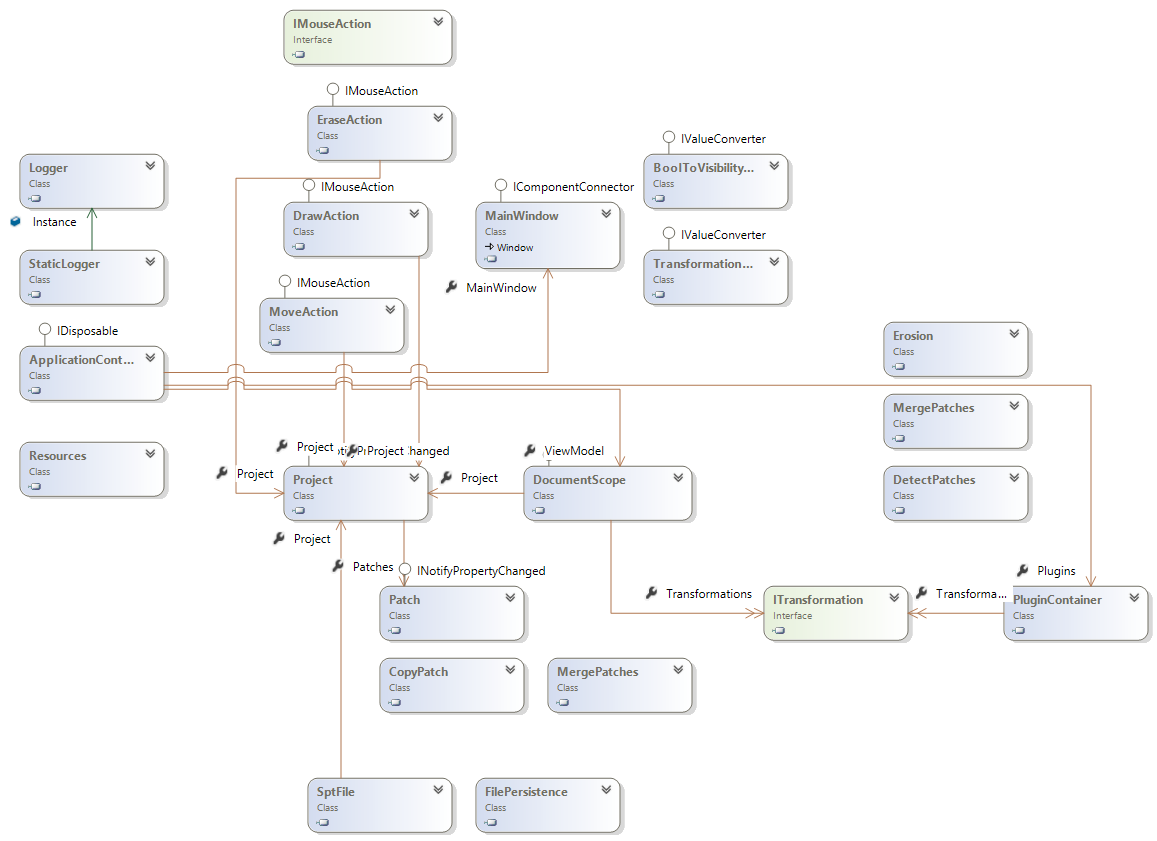
Az intuitivitás érdekében minél kifejezőbb grafikus ikonok kiválasztására törekedtünk, és arra az esetre is felkészültünk, ha a felhasználó az igyekezetünk ellenére mégsem tudja kitalálni, milyen funkcióra alkalmas az ikon: az egeret bármelyikük fölé mozgatva tooltip jelenik meg.

A program lokalizációra, azaz többféle nyelv támogatására alkalmas. Jelenleg az amerikai angol, illetve a magyar nyelv választható, de igény esetén gyorsan tudnánk még több nyelvvel bővíteni az alkalmazást.

# Megvalósítás

## Osztálydiagram

Az alábbi osztálydiagramot Visual Studio-val generáltuk. Látható, hogy viszonylag nagyszámú, 23 db osztályból áll a projekt forráskódja.

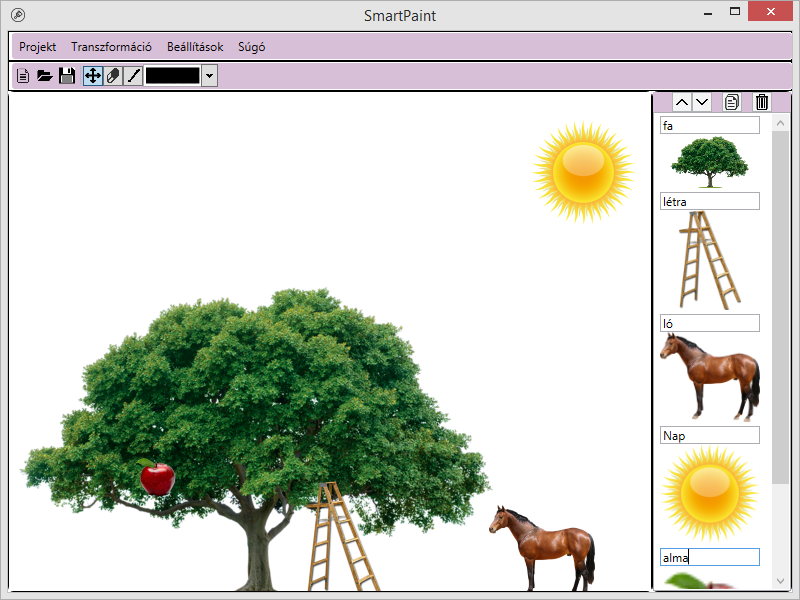


4. ábra: Az alkalmazás osztálydiagramja

## Képernyőképek

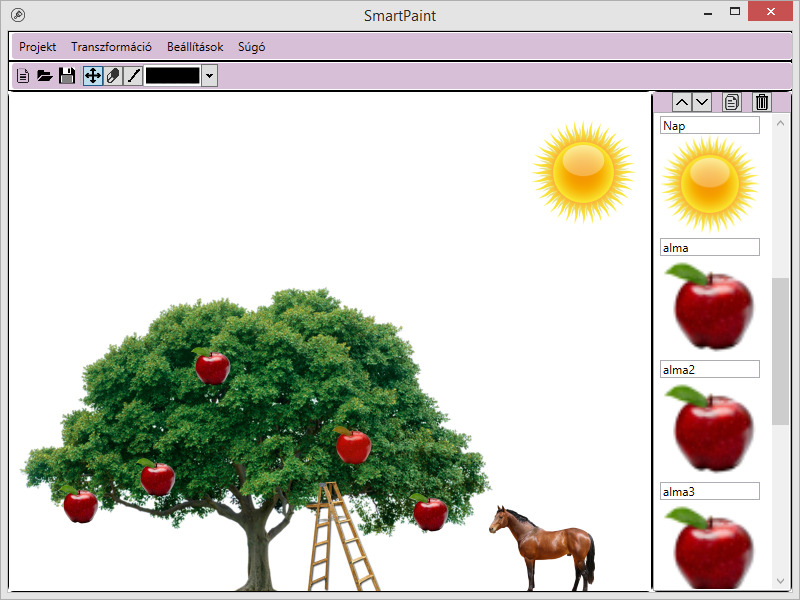
### Alapvető műveletek alakzatokkal

Az alábbi képen látható, hogy a programba betöltött különböző alakzatok tetszőlegesen mozgathatóak a vásznon, és a kirajzolási sorrend is állítható a nyilak segítségével. A képeket természetesen az átlátszósági információ figyelembevételével rajzolja ki a program. Az alakzatoknak tetszőleges nevet lehet adni. Az alakzatot a kuka ikonnal ki is lehet törölni, ha a felhasználó úgy dönt, hogy mégsincs rá szüksége.



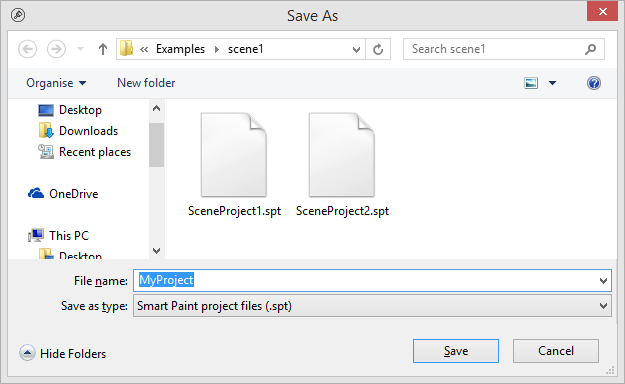
5. ábra: Több importált kép mozgatásával szép kompozíció alakítható ki

Ha a felhasználó egy alakzatból egynél többet szeretne, például a képen látható fát sok-sok almával kívánja díszíteni, akkor a többszöri importáláson kívül a másolásra is lehetősége van.



6. ábra: Másolás működése

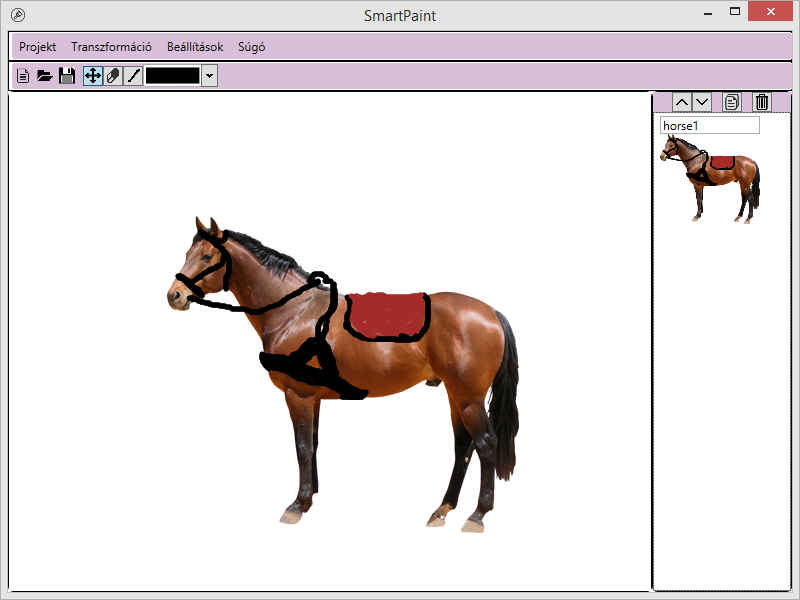
Ha a felhasználó elégedett az elkészített képpel, akkor .png képfájlba exportálhatja. Ha később is dolgozni kíván rajta, akkor egy .spt kiterjesztésű SmartPaint projektfájlba mentheti a munkáját. Ezt a fájlt betöltve ugyanott folytathatja a munkát, ahol abbahagyta.



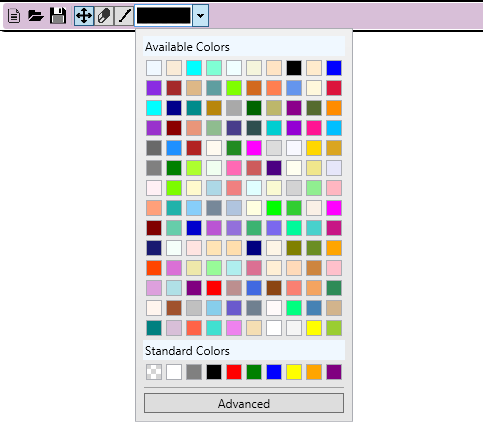
7. ábra: Projekt mentését végző dialógusablak

### Ecset és radír

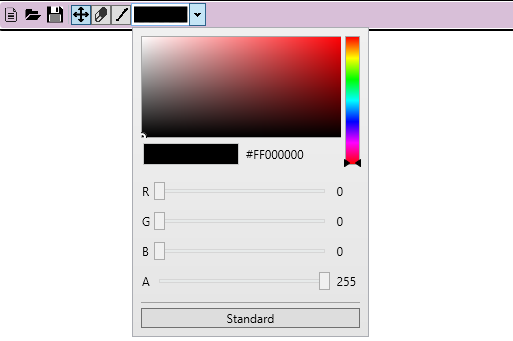
A beimportált képekre tetszőlegesen kiválasztott színekkel rajzolhatunk az ecset segítségével. Lehetőségünk van előre megadott kész színekkel dolgozni, de a haladó felhasználók bármilyen színt kiválaszthatnak a spektrumon, sőt, akár RGB kóddal is megadhatják.



8. ábra: Rajzolás tetszőleges színekkel

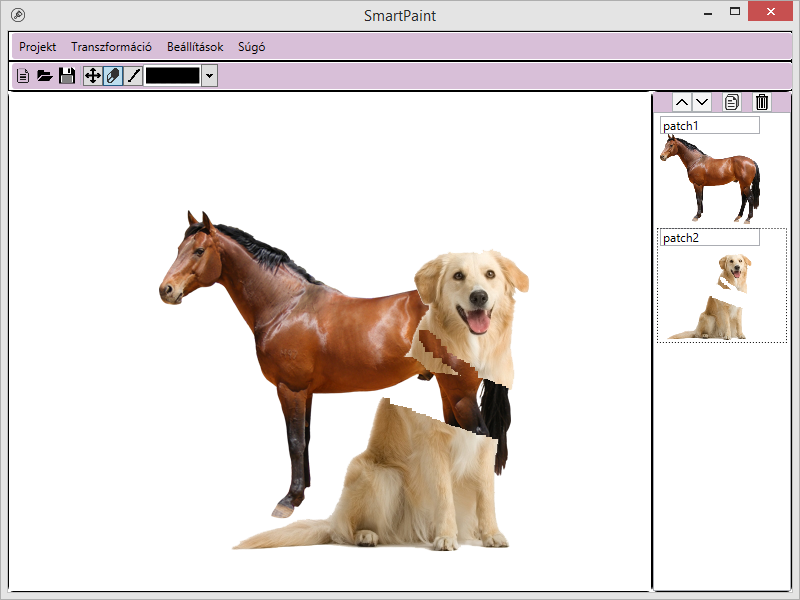


9. ábra: Színválasztó komponens standard nézete, ahol rengeteg előre definiált szín közül választhatunk



10. ábra: Színválasztó komponens haladó nézete, ahol bármilyen színt elő lehet állítani

A rajzoláson kívül radírozásra is van lehetőség, ami a színes pixelt teljesen átlátszóvá alakítja át.



11. ábra:Radír használata

### Transzformációk

A SmartPaint alapvetően három beépített transzformációt tartalmaz: a különálló foltokra bontást, az összeolvasztást és az eróziót. A bontás, ahogyan a neve is sugallja, a képet több külön alakzatra bontja úgy, hogy a bontás után minden alakzat összefüggő legyen. Az összeolvasztás lényege, hogy két alakzat eggyé válik. A lent bemutatott példában a különálló fa és létra összeolvasztásával jön létre a fa+létra nevű alakzat.

|  |  |
| --- | --- |
|  |  |

12. ábra: Összeolvasztás működés: létrából és fából ’fa+létra’ lett

Az erózió egy olyan transzformáció, ami a kiválasztott alakzatok szélső pixeleit kivágja.

|  |  |
| --- | --- |
|  |  |

13. ábra: Erózió működése (a képen 4 iteráció eredménye látható)

Az alkalmazás érdekessége, hogy a transzformációkat dinamikusan tölti be egy Plugins nevű mappából. Ez azt jelenti, hogy bárki írhat saját transzformációt. Ha ez a megfelelő interfészt implementálja, és dll-ként behelyezik a Plugins mappába, akkor az egyéni transzformáció is ugyanúgy használható, mint a beépítettek. (Ehhez hasonló módon működik a Paint.NET nevű népszerű rajzprogram pluginokkal való bővítése is.)

# Összefoglalás

## Kihívások, tanulságok

A fejlesztés során mélyebben megismertük a .NET binding sokrétűségét, és a WPF felhasználói felület gazdag funkcionalitását. Néhány bonyolultabb esetben kihívást jelentett a modell és a nézet összekapcsolásakor a változásértesítés megvalósítása. Ezen kívül az eróziót előállító képfeldolgozó algoritmust is megismertük.

Kihívást jelentett a radírozás megvalósítása amiatt, hogy a WPF rajzolóutasításai alapvetően nincsenek erre felkészítve. A radírozást ezért végül úgy kellett megoldanunk, hogy tömb szinten töröltük a megfelelő pixeleket, miután a képet WriteableBitmappá alakítottuk át.

Az alkalmazás futása során számos konverzióra van szükség BitmapImage, WriteableBitmap és RenderTargetBitmap között, ugyanis a WPF különböző utasításai különböző képformátumokon operálnak.

A fentiek tanulsága, hogy a WPF ugyan a legtöbb esetben nagyon egyszerűen és hatékonyan használható UI fejlesztése, sajnos nem rajzolóprogram készítésére van kitalálva.

## Továbbfejlesztési lehetőségek

A SmartPaintet természetesen rengeteg hasznos funkcióval lehetne még bővíteni, aminek csak a fejlesztőcsapat ideje szab határt. Néhány lehetőség a teljesség igénye nélkül:

* Több transzformáció
* Több támogatott nyelv
* Többféle méretű, alakú, illetve stílusú ecset és radír
* Átméretezhető rajzvászon
* Olyan felhasználói élményt javító funkciók, mint például visszavonás, kilépés előtt változások mentésének megkérdezése
* Professzionálisabb megjelenésű felhasználói felület, egyedi ablakkal és témával

# Telepítési útmutató

Mivel az alkalmazás hordozható (portable), nem szükséges telepíteni. Csak ki kell csomagolni a mellékelt .zip archívumot, és ezután a SmartPaint.exe alkalmazást már el is lehet indítani. Az alkalmazás nyelve az első indításkor angol, de a Settings → Language menüben a Hungarian-t választva átállítható magyarra. A változás érvénybe lépéséhez a programot újra kell indítani, és ezt követően a nyelv tartósan magyar marad, de tetszés szerint bármikor visszaállítható angolra ugyanezzel a módszerrel.

Ha a felhasználó esetleg saját transzformációval kívánja bővíteni a programot, akkor a .dll fájlt az alkalmazás Plugins mappájába kell helyeznie, és a következő indításkor a Transformations menüben automatikusan megjelenik az új funkció.

A futtatáshoz szükséges minimális rendszerkövetelmények az alábbiak:

* Operációs rendszer: Microsoft Windows 7,8, vagy egyéb, .NET 4.0-t tartalmazó rendszer
* Legalább 10 MB szabad lemezterület
* Legalább 2 GB RAM memória
* 1 GHz-es, vagy jobb processzor

# Hivatkozások

A fejlesztés során felhasznált eszközök listája:

* Microsoft Visual Studio Ultimate 2012: A célunknak legjobban megfelelő fejlesztőeszköz.
* WPF Extended Toolkit: Egyedi WPF komponenseket tartalmazó eszköztár. A SmartPAint felhasználói felületén látható színválasztó komponenst innen vettük.
* DotNetZip: A program által használt .spt projektformátumba való mentéshez használtuk fel. A .spt tulajdonképpen egy becsomagolt .zip archívum, amit a DotNetZippel állítottunk elő.
* Git: A közös munkát nagymértékben megkönnyítette a Git verziókezelő rendszer.
* GitHub: A forráskód online hosztolásához a GitHubot választottuk.
* SourceTree: Grafikus Git felhasználói felület, melynek használata jóval kényelmesebb, mint a Git parancssoré.
* Microsoft Visio: Dokumentációs eszköz, amit az osztálydiagram és use-case diagram elkészítéséhez használtunk.