**WPF - Windows Presentation Foundation**

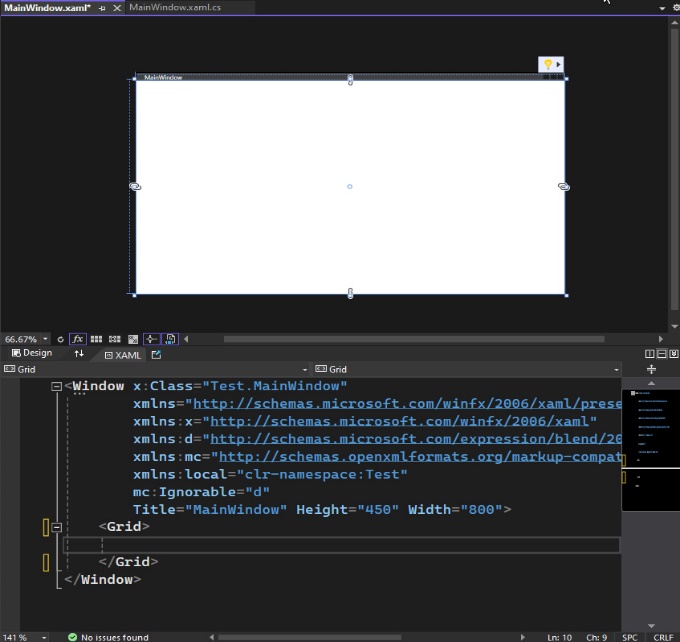
WPF je korisnički interfejs koji nam služi za razvijanje aplikacija za **desktop**. Nastao je 2006 godine kao zamena za Windows forms, iako se i dan danas oba i dalje koriste.

|  |  |
| --- | --- |
| Razlika između WinForms-a i WPF | |
| WinForms | WPF |
| Koristi se za jednostavniji layout | Koristi se za složeniji layout |
| Manje fleksibilan | Više fleksibilan |
| Koristi **raster-based graphics system,** što ga limitira na prostiju u manje kvalitetnu grafku | Koristi **vector-based graphics** sistem ,što omogućava kvalitetniju i više skalabilnu grafiku, kojom je lakše upravljati |
| Za prostije aplikacije | Za složenije aplikacije, optimalnije za kodiranje igrica nego WinForms |
| Može da se koristi i na Windows-u, Linux-u, Mac OS | Koristi se primarno za Windows |
| Koristi tradicionalniji pristup za Layout, Forme i kontrole | Koristi XAML za layout |
| Nema **DataBinding** | Ima **DataBinding** |

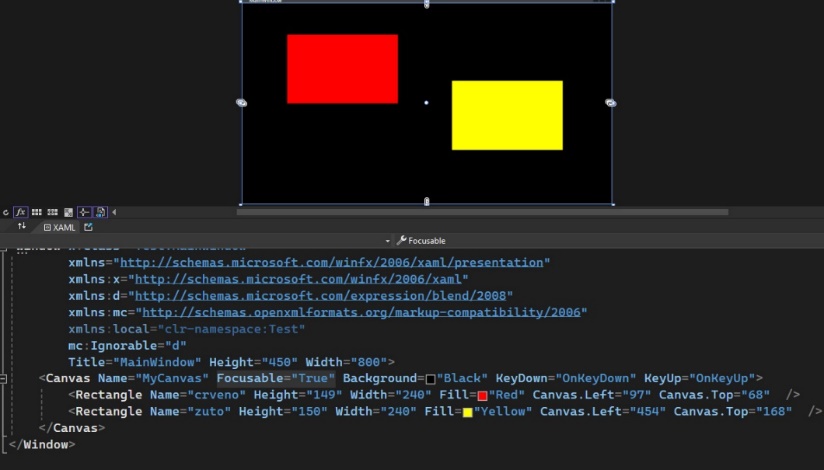
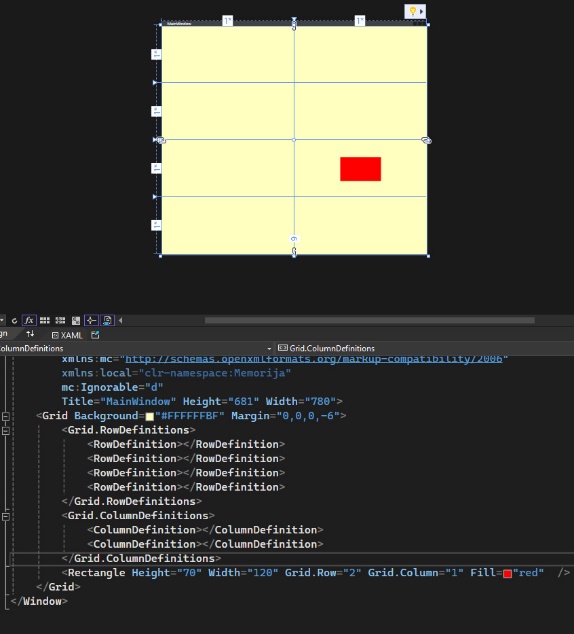
WPF je ugrađen u .NET pa može da se koristi u VisualStudiu. Umesto pozivanja konzole da interaguju sa korisnikom, koristimo **XAML (Extensible Application Markup Language).**

**Upoznavanje sa XAML-om**

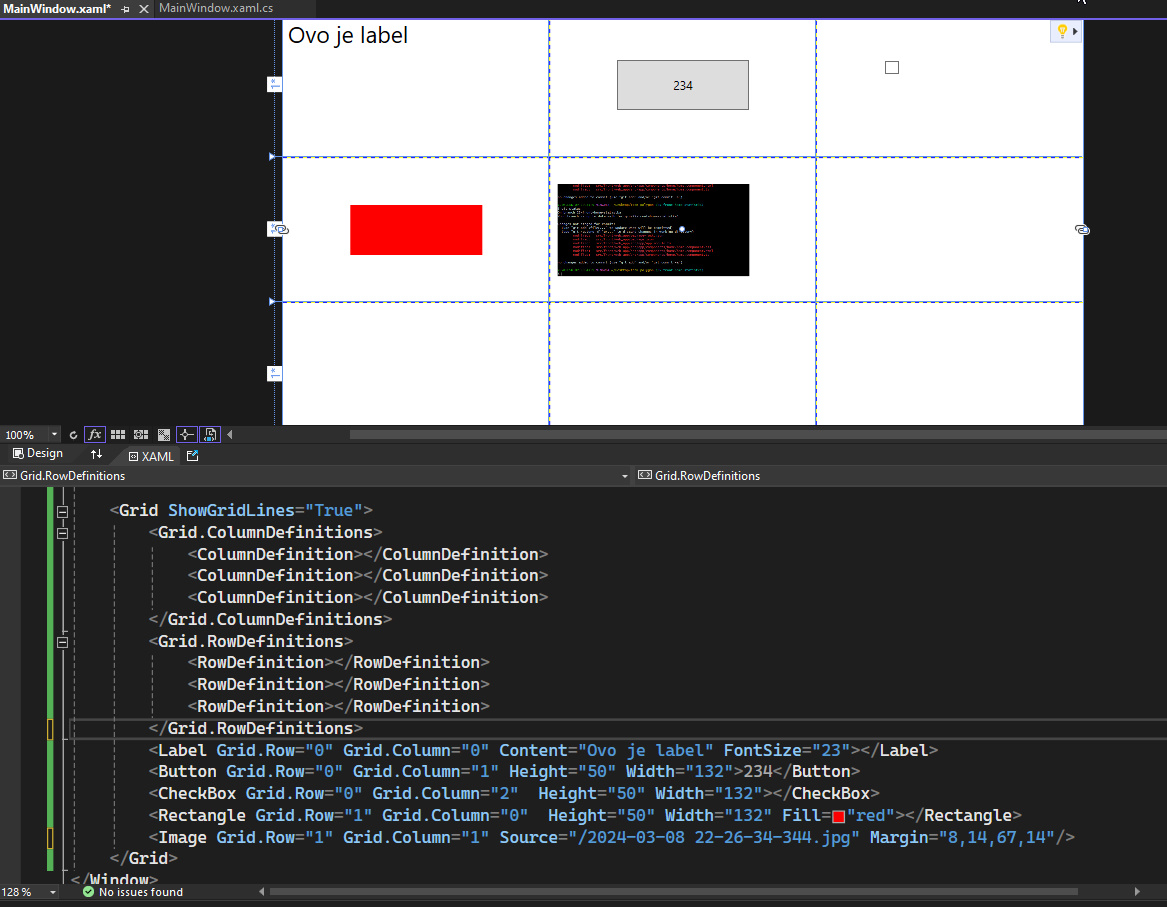
Kada otvorimo novi wpf projekat imamo **MainWindow.xaml** sa **MainWindow.cs** i **App.xaml** sa **App.cs,** nam služi za komplikovanije projekte za skladištenje resursa, tako da nam to trenutno nije potrebno. Na **MainWindow.xaml** možemo da vidimo da je veoma sličan winFormi, glavna razlika je što nam je na stranici za dizajn aplikacije osim toolbox-a koji nam služi za ubacivanje komponenti u projekat, dat i XAML prozor, koji predstavljana pismenu reprezentaciju svega što vidimo na prozoru iznad, **svaku promenu koju učinimo u XAML će se odraziti na formi iznad, i obrnuto**. Mi preko XAML -a možemo da ubacujemo komponente dizajn, u wpf-u ne treba da se ubacuju preko ToolBox-a! Ovim se gubi snaga wpf-a, i projekat postaje bliži winFormama. Ubacivanje preko XAML-a uopšte nije komplikovano.

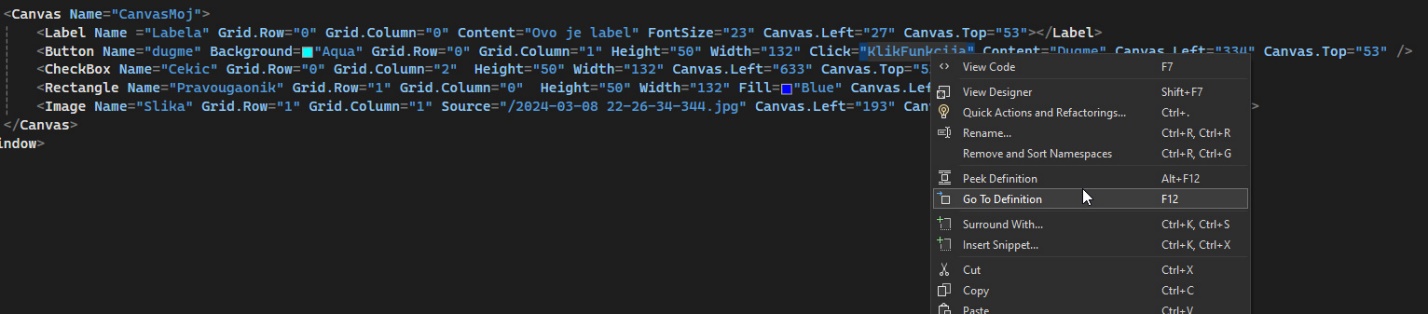


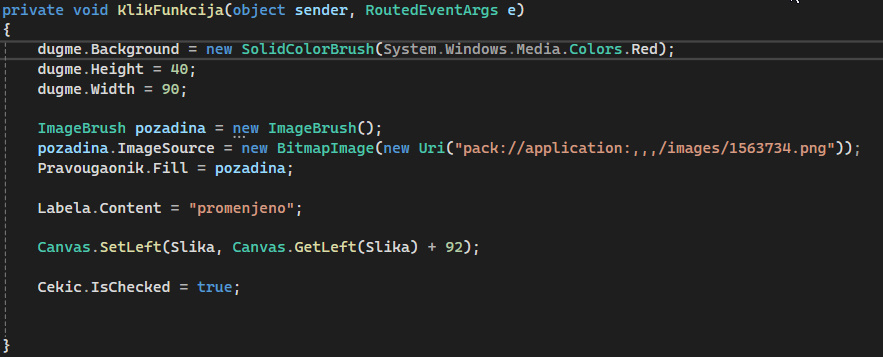
XAML je veoma sličan HTML-u . Elementi su prikazani otvarajućim tagovima <Window> zajedno za zatvarajući tagovima </Window> , sve između predstavljaju elemente unutar elementa <Window></Window>. Unutar otvorenog taga elementa je moguće upisati atribute istog ; u slučaju <Window> imamo **Title** – naslov i **Height** I **Width** – dimenzije našeg projekta; još jedan bitan atribut je **name**, koji nam daje ime reference na objekat elementa. služi da bismo kasnije mogli da rukujemo istim. Da bismo dizajnirali našu igricu potrebno je da sve elemente obuhvatimo određenim elementom za dizajniranje ,a to su **Grid** ili **Canvas** . **Grid** se više koristi za pravilan raspored elemenata, podržava ređanje elemenata u kolonama I redovima, a **Canvas** se koristi za proizvoljno određivanje lokacije. Pomoću **Grid**.**RowDefinitions** I **Grid.ColumnDefinitions-a** možemo da definišemo broj kolona I redova u Gridu tako što dodajem onoliko **Column/RowDefinition** elemenata Koliko ima redova/kolona. Lokaciju elementa unutar Canvas-a određujemo atributima, **Canvas.Left** – udaljenost od leve granice Canvas-a I **Canvas.Top** – udaljenost od gornje granice Canvas-a (može I Canvas.Right I Canvas.Bottom, ali prethodno je optimalnije), ukoliko koristimo tastaturu za rad naše aplikacije (pomeramo lika na pritisak na dugme) obavezno staviti Focusable="True" u canvas-u da bi aplikacija primila događaj pritiska na tastaturu.



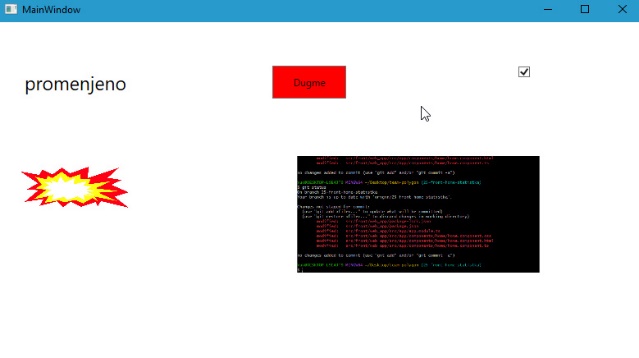
**Bitniji elementi u XAML-u**

1. **Window** - Prozor najvišeg nivoa koji sadrži druge elemente.
2. **Grid** - Pruža raspored u obliku tabele za organizaciju elemenata u redove i kolone.
3. **Canvas** - Pruža raspored apsolutnog pozicioniranja, omogućava precizno postavljanje elemenata.
4. **Rectangle** – Pravougaonik, u kreiranja igrica se skoro svaki element predstavlja u vidu pravougaonika, na koji se kasnije dodeljuje slika kao pozadina
5. **Label** – Označava tekst koji ne može da se menja.
6. **Image** – Nećemo ovo da koristimo, da bi smo ubacili sliku prosto ćemo da ubacimo pravougaonik i kasnije staviti sliku na isti u .cs fajlu.
7. **Button,TextBox,ComboBox, CheckBox, RadioButton** – Upoznali ste se sa ovim kod winForme.

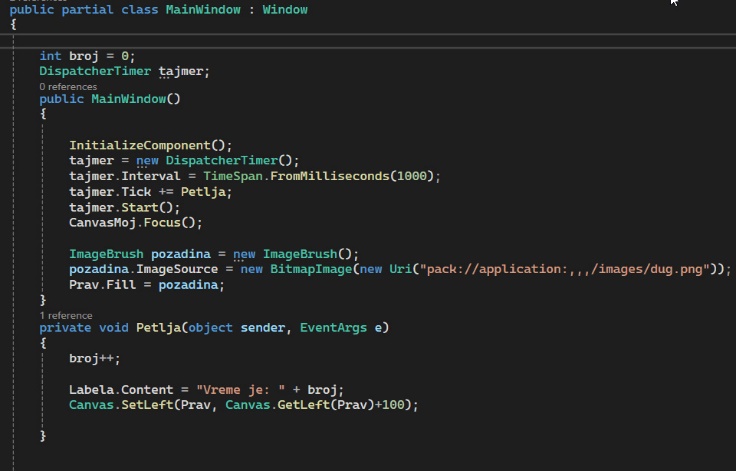
**Definisanje događaja u WPF-u**

****Veoma slično kao u WinFormi, da bi smo definisali događaje poput **onClick, MouseIn, MouseOut, keyDown, keyUp**, potrebno je u otvarajućem tagu elementa na koji treba da se izvrši događaj dodati tip, i naziv funkcije koja rukovodi događajem, da bi smo najlakši otišli do funkcije gde definišemo rezultat događaja potrebno je da selektujemo naziv funkcije, desni klik, i izaberemo opciju go to definition, ovo će automatski napraviti funkciju gde možemo da odlučimo šta se dešava prilikom događaja. Na primeru vidimo da pritiskom na dugme možemo da promenimo boju I dimenzije istog, sliku pozadine pravougaonika levo, lokaciju slike u sredini, tekst na label-u i status checkbox-a / čekiramo ga.

Da bi smo dodali sliku pravouglu moramo da definišemo objekat **ImageBrush** koji nam omogućava da ‘’obojimo’’ pravougaonik u proizvoljnu sliku koju biramo pomoću **BitImage** gde definišemo putanju slike (slika mora bude u projektu da bi ovaj način funkcionisao). Ovako se ubacuju slike pravilno u wpf-u zbog fleksibilnosti – mogu da se menjaju slike nekog elementa po želji veoma lako.

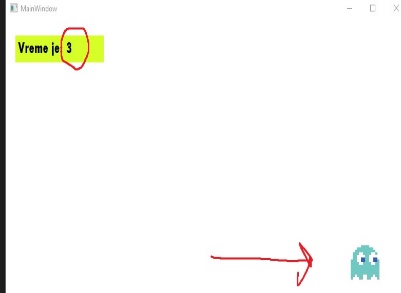
****

**DispatcherTimer**

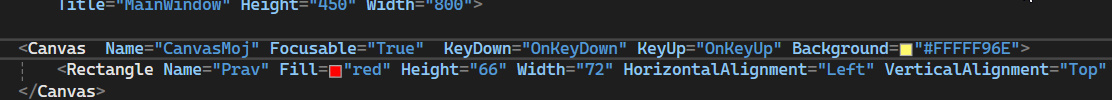
Jedna od najbitnijih novina u WPF-u je **DispatcherTimer**. Ova klasa se često koristi za izvršavanje određenog koda u definisanom intervalu, idealan za kreiranje tajmera kao i za pomeranje određenog elementa u aplikaciji bez interakcije korisnika. Da bismo implementirali tajmer potrebno je da u klasi MainWindow definišemo atribut klase **DispatcherTimer** ,onda u konstruktoru definišemo određeni interval između 2 Tik događaja (interval definišemo u milisekundama), Nakon toga je potrebno da dodelimo događaju **Tick** određenu metodu koju želimo da se izvrši prilikom njegovog podizanja, u našem slučaju to je povećanje vrednosti tajmera i pomeranje pravougla dole za proizvoljnu vrednost. Obaveznu pozvari metodu .Start() da bi se tajmer pokrenuo.

A video game with a green and yellow ghost

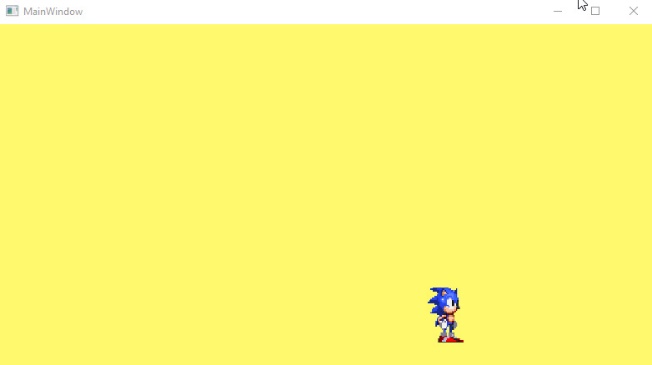
Description automatically generated

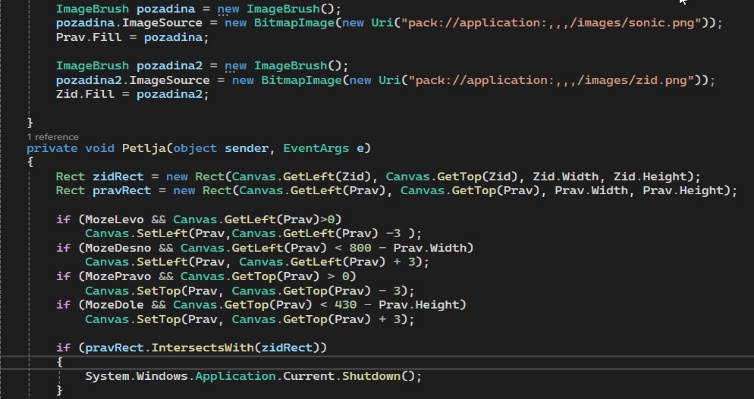
Upozorenje: iako DispatcherTimer ne može da porani, može da mu se desi da zakasni. Bolje je koristi odvojene tajmere za pomeranje lika I tajmer

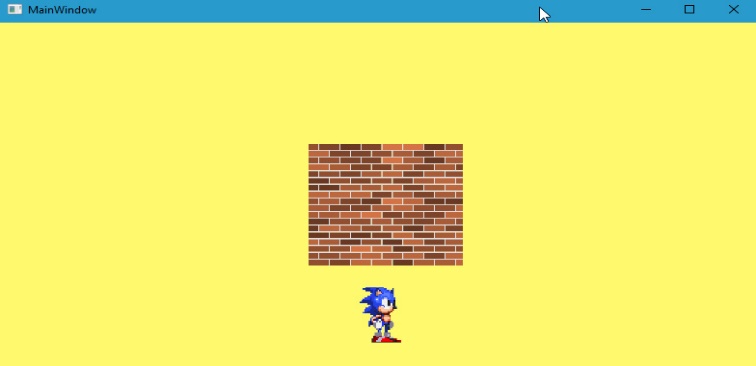
**Događaji vezani za tastaturu**

Da bi smo uopšte mogli da koristimo tastere u aplikaciji potrebno je da u XAML u canvas-u stavimo Focusable="True", takođe u konstruktoru MainWindow-a stavimo: CanvasMoj.Focus().

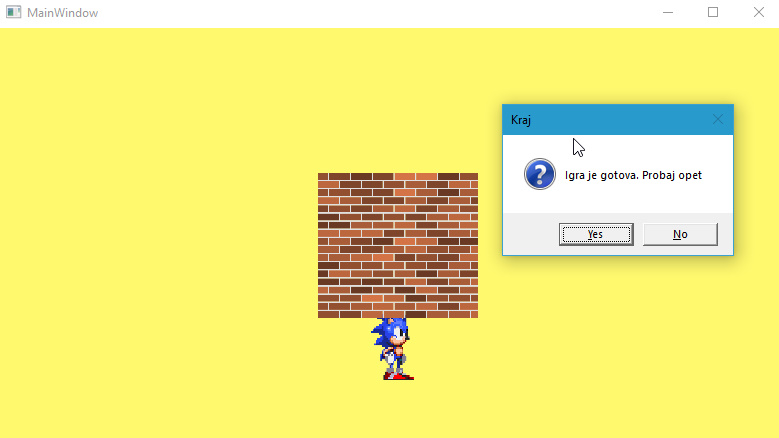
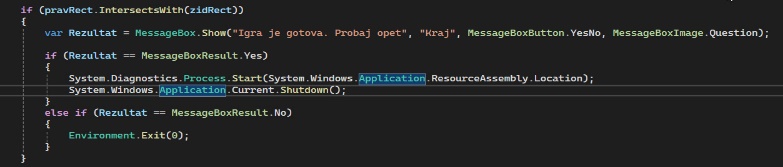
Da bi smo na najoptimalniji način implementirali kretanje lika na pritisak na taster potrebno je da izvršimo implementaciju korišćenjem **DispatcherTimer-a**, Potrebno je definisati dva **bool**-a za levu i desnu stranu (i za pravo i nazad, ako i to implementirano) i staviti da je podrazumevana vrednost **false**. U funkciji koja se dodeljuje događaju Tick u timer-u stavljamo da se element koji kontrolišemo, pomeri za određenu vrednost u smer kojem je bool namešten da **True**, ovim se postiže iluzija While petlje u smislu ’’dok je taster pritisnut, pomeraj’’. Potrebno je da omogućimo da na klik na taster (KeyDown) se prebaci bool za odgovarajući smer na **true**, suprotno - kad pustimo taster (KeyUp) bool se prebacuje na **false**. Ono što name je bitno je da sprečimo da element kojim manipulišemo napusti granice prozora, ovo prosto postižemo tako što tako što u



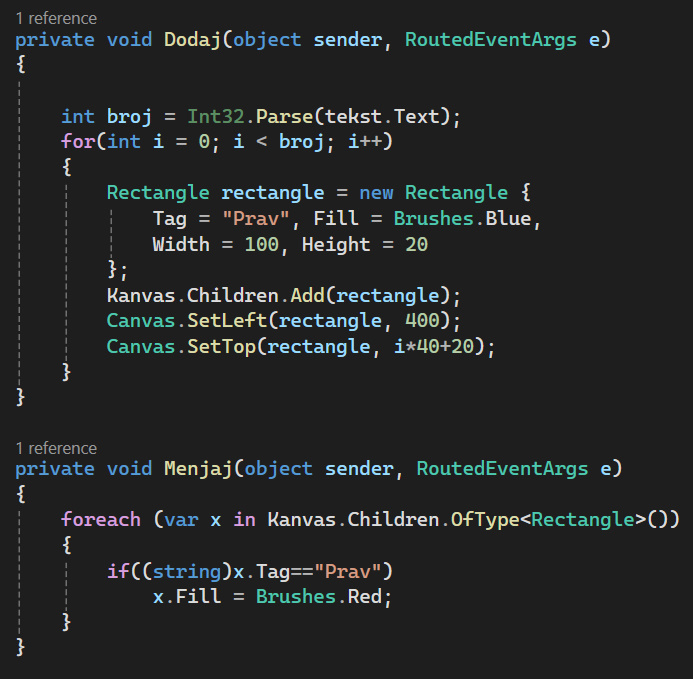
**Interakcija sa drugim elementima**

****WPF Nam omogućava prostu već definisanu implementaciju interakcije, već je definisana metoda sa bool kao povratnom vrednošću **Element1.IntersectsWith(Element2).** Neophodno je da stvorimo nove Rect objekte koji imaju vrednosti elemenata koji sudaraju. (Rectangle definisan u XAML nemaju metodu **IntersectsWith**). Kad želimo da se nešto desi prilikom ’’sudaranja’’ sa elementom, samo treba da napišemo nakon if zagrade. U sledećem primeru ako se lik sudari sa zidom, gasi se aplikacija.

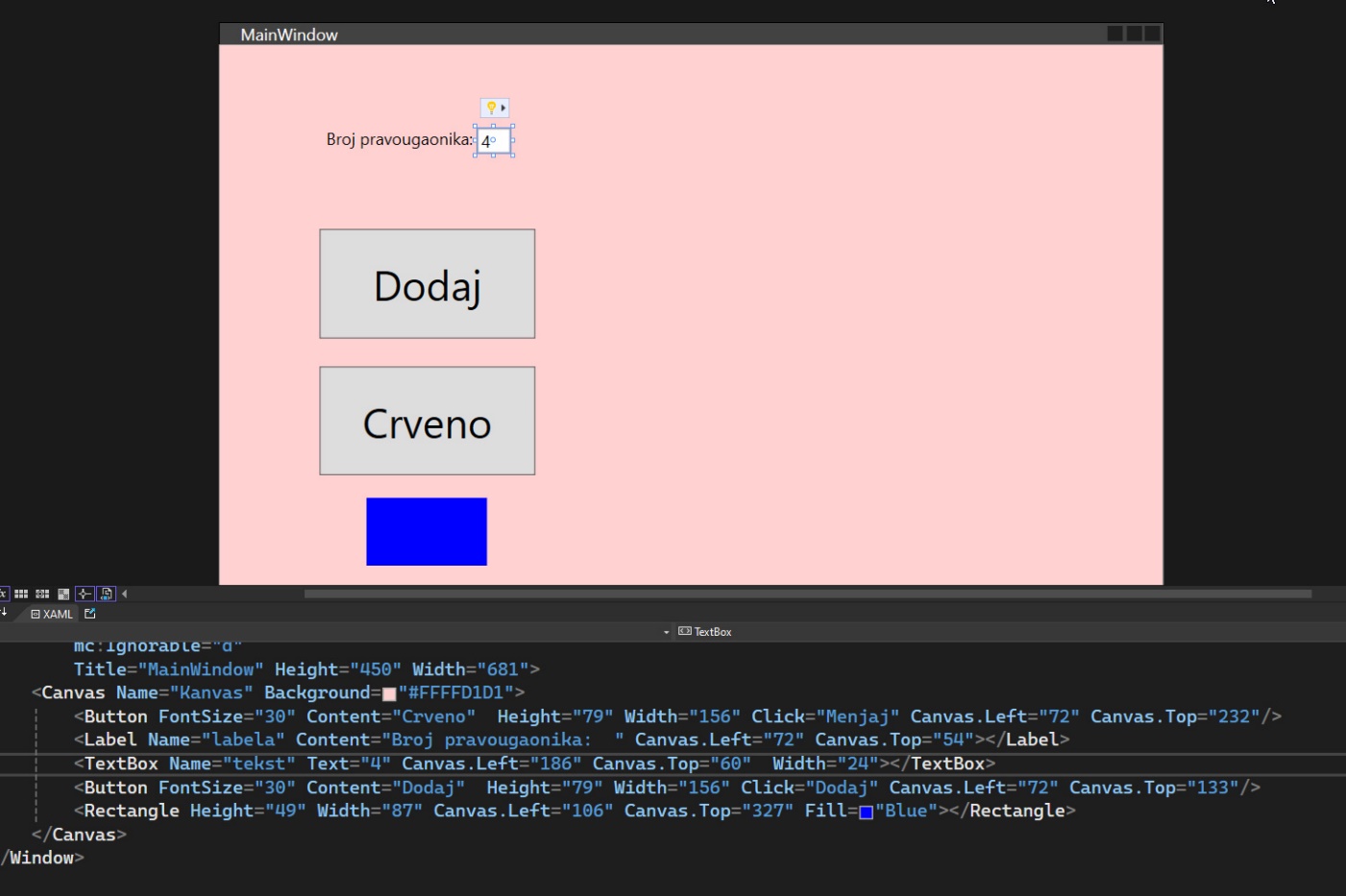
**Popup Window**

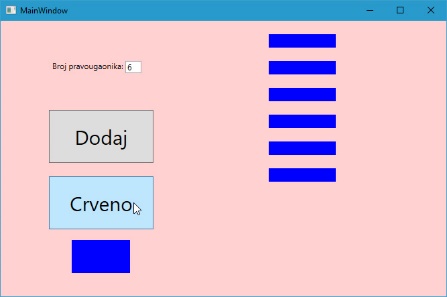
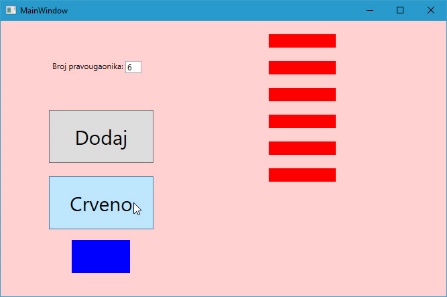
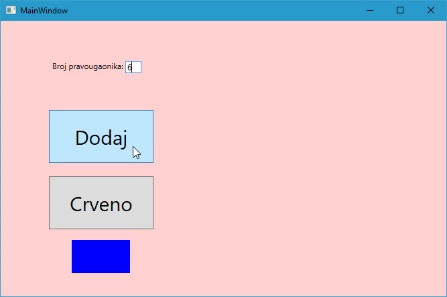
Nadovezaćemo se na prethodnu sekciju. Umesto da se ugasi aplikacija prilikom sudara sa zidom otvori se prozor gde birama opciju ’’da’’ ako želimo da započnemo aplikaciju ponovo, ili ’’Ne’’ ukoliko smo završili.

**Ubacivanje elementa preko .cs**

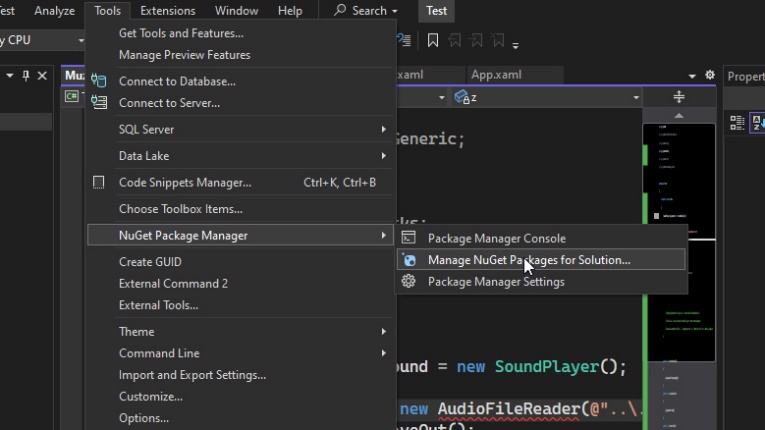
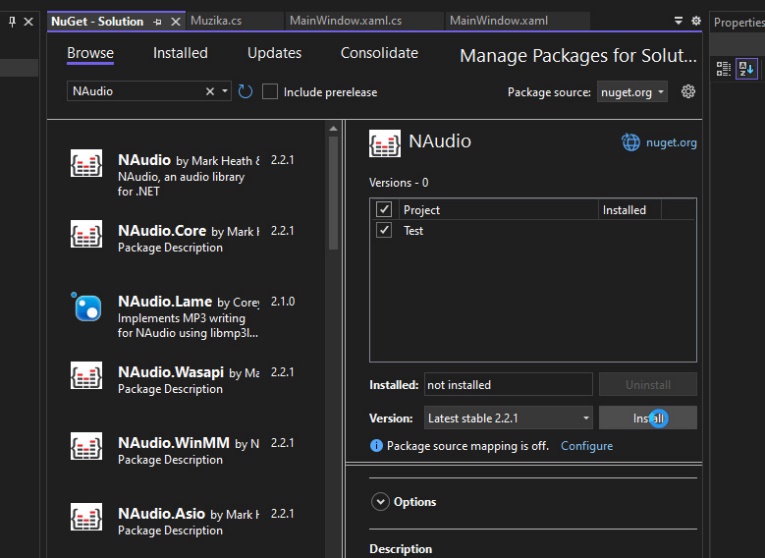
****Postoje slučajevi gde moramo da ubacujemo elemente van XAML-a npr. imamo dugme koji pravi određeni broj novih pravougaonika. Da bi smo to uradili potrebno je da kreiramo određeni broj objekata odredimo im visinu,širinu, i poziciju (da se ne stvore svi objekti da istom mestu) i veoma je važno da zapišemo **tag** svih elemenata, ukoliko želimo da kasnije vršimo neke operacije nad tim pravougaonicima koje smo napravili upravo. Kada stvorimo objekat potrebno je da ga dodamo u određeni element za dizajniranje (Grid,Canvas) kao ’’dete’’ tog elementa.

Ako želimo vršimo neke operacije nad novo stvorenim pravougaonicima potrebno je da prođemo kroz sve pravougaonike koje pripadaju Canvas-u i izvršimo operaciju samo nad onim koji imaju tag koji smo odabrali.

****



**(Bonus) Ubacivanje muzike/zvuka**

Za ubacivanje zvuka u aplikaciju koristimo klasu **SoundPlayer** ,definišemo lokaciju zvučnog fajla i izaberemo hoćemo li pustiti da se zvuk pušta jednom **SoundPlayer**.**Play()**, ili da se ponavlja **SoundPlayer**.**PlayLooping()**, Problem kod ovog rešenja je što samo jedan zvuk sme da se pušta u isto vreme (ako se pozove metoda Play/Playlooping drugog **SoundPlayer-a** dok prvi još uvek pušta muziku, prvi će se ugasiti.) Ovo rešavamo korišćenjem **WaveStream-a** , ali moramo prvo da instaliramo **NAudio.Wave** paket. Pomoću **WaveStream-a** možemo da pustimo više zvučnih fajlova u isto vreme. Ukoliko se neki zvuk ponavlja više puta za redom, možemo da pri događaju koji aktivira zvuk zaustavimo zvuk ako nije već završen, vratimo ga na početak i pustimo ga ponovo. Ekstenzija za zvučni fajl koji koristimo bi trebalo da bude **.wav**

