Windows Phone App

Guild Wars 2 Event Timer App

Dit verslag bevat uitleg over de applicatie die ontwikkeld werd voor het vak Mobiele Apps. Er wordt uitgelegd wat Guild Wars 2 is, wat de applicatie doet en welke functie elk stukje code heeft.

2015

Sarah Hansen

2ICT2

29-5-2015

Windows Phone Applicatie

Inhoud

[Inleiding 2](#_Toc420623471)

[Wat is Guild Wars 2? 3](#_Toc420623472)

[Wat doet mijn applicatie? 3](#_Toc420623473)

[Code van de applicatie 4](#_Toc420623474)

[MainPage.xaml 4](#_Toc420623475)

[MainPage.xaml.cs 6](#_Toc420623476)

[MainViewModel.cs 7](#_Toc420623477)

[ItemViewModel.cs 12](#_Toc420623478)

[App.xaml.cs 13](#_Toc420623479)

[FileIO.cs 14](#_Toc420623480)

[Afbeeldingen 16](#_Toc420623481)

[Achtergrond afbeelding 16](#_Toc420623482)

[Screenshots 17](#_Toc420623483)

[Up pivotItem 17](#_Toc420623484)

[2nd pivotItem 18](#_Toc420623485)

[3rd pivotItem 19](#_Toc420623486)

[Slot 20](#_Toc420623487)

# Inleiding

Als 2e jaar ICT studenten kregen we de opdracht om een applicatie te maken voor Windows Phone. Ik heb voor een timer applicatie gekozen die om de 15minuten nieuwe informatie toont die uit een xml gelezen wordt. In dit verslag zal ik zeer oppervlakkig uitleggen wat Guild Wars 2 is. Een kleine uitleg van de applicatie voordat ik de gehele code zal bespreken. Elke klasse zal ik zo goed mogelijk proberen uit te leggen wat de code per methode doet.

Als laatste zal ik nog 3 screenshots van mijn applicatie invoegen, een voor elk pivotItem, en de achtergrond afbeelding die ik voor mijn applicatie gebruikt heb. In de repository zelf heb ik een map met een screenshot van alle events die kunnen verschijnen toegevoegd en de afbeeldingen van elke event die in gebruik in mijn applicatie.

# Wat is Guild Wars 2?

Guild wars 2 is een MMORPG, Massively multiplayer online role-playing game, of simpel gezegd een online game waar je vrij kan spelen met andere mensen. Het is ontwikkeld door het Amerikaans bedrijf ArenaNet en word uitgegeven door NCSoft. Mensen hebben mogelijkheid om tegen elkaar te vechten, tegen computer genereerde vijanden te vechten of in grote georganiseerde groepen tegen elkaar te strijden.

Om het kwartier kan je ook deelnemen aan een grotere activiteit waar iedereen voor samen zal komen. Dit heet “world bosses”. Over de hele game zal dan op een vaste plaats een monster verschijnen. De tijd dat elke monster verschijnt, is gebonden aan de UTC tijd en zal niet meer veranderen met onze zomer- of wintertijden. Ik heb deze informatie van deze events verzameld en neergeschreven in een xml bestand.

# Wat doet mijn applicatie?

De informatie over het spel en zijn events die ik heb opgeslagen in een xml-bestand wordt online gezet via github. Mijn applicatie zal dit bestand inlezen en in een lijst verwerken. Een constante timer zal om de seconden de pagina’s vernieuwen. Voor de 2e en 3e pivot pagina verandert er slechts om de 15min iets. De 1e pivot en hoofdpagina zal een timer laten lopen die aftelt tot het moment dat het event begint, daarna blijft de boss nog even zichtbaar om de gebruiker nog de kans te geven om te kijken als het events misschien nog beschikbaar is.

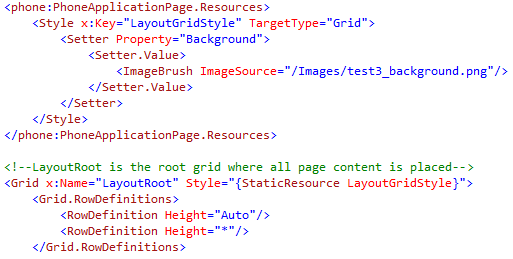
Intern wordt in de klasse MainViewModel het online bestand ingelezen en met een observableCollection Items gebonden die de structuur van het viewmodel ItemViewModel heeft. In de MainViewModel kan je dan een waarde ophalen aan de hand van index en naam. Bijvoorbeeld this.Items[0].BossName geeft de naam van de eerste bos in de xml lijst terug.

De klasse MainViewModel zal de huidige UTC tijd van het gsm vergelijken met de UTC tijden in de xml lijst. Als er een tijd in de komende 10min overeenkomt zal deze getoond worden en de 2 daaropvolgende events ook. De observableCollection Items zal voor het gevonden index meegestuurd worden naar de MainPage waar de data gebonden wordt aan de waarden in Items. Voor elke pivot zal deze waarden dus uiteraard verschillen.

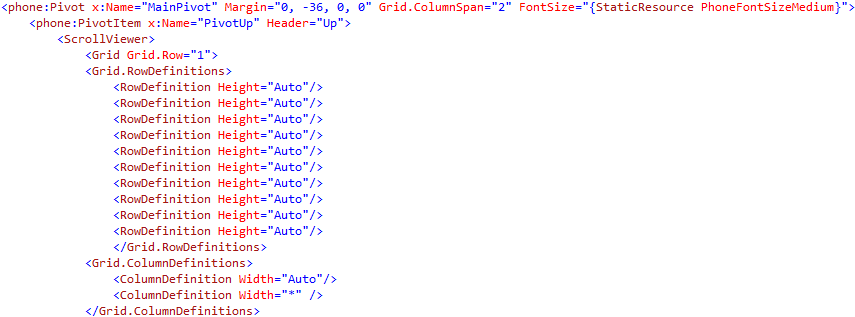
# Code van de applicatie

## MainPage.xaml

De MainPage.xaml is de grafische klasse die bepaald hoe de applicatie zal uitzien en waar de waarden uit de lijst worden ingevuld door databinding. In de code hieronder voeg ik een achtergrond afbeelding toe die zich over heel de pagina en alle pivots verspreid. Ik heb de afbeelding aan een style gekoppeld en die meegestuurd met hoofd Grid.

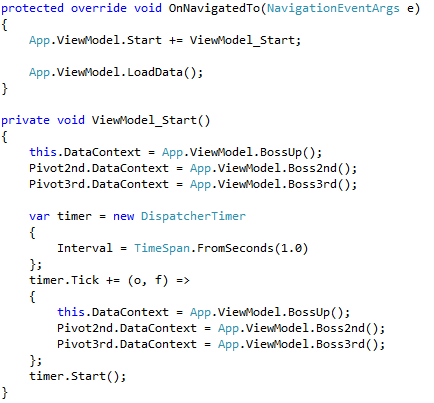


Vervolgens maken we de 3 verschillende pivots aan. De pivotItems krijgen een naam mee zodat ze makkelijker te binden zijn met de data. De homepage krijgt de naam PivotUp. De twee daarop volgende pivotItems krijgen respectievelijk Pivot2nd en Pivot3rd. De code voor alle pivotItems is exact hetzelfde, inclusief de databinding namen. De optie <scrollviewer> staat toe dat gebruiker de pagina verticaal kan bewegen. De grid maakt het mogelijk om gemakkelijk de textblocks te sorteren. De opbouw voor een pivotItem is 2 kolommen en 10 rijen.



## MainPage.xaml.cs

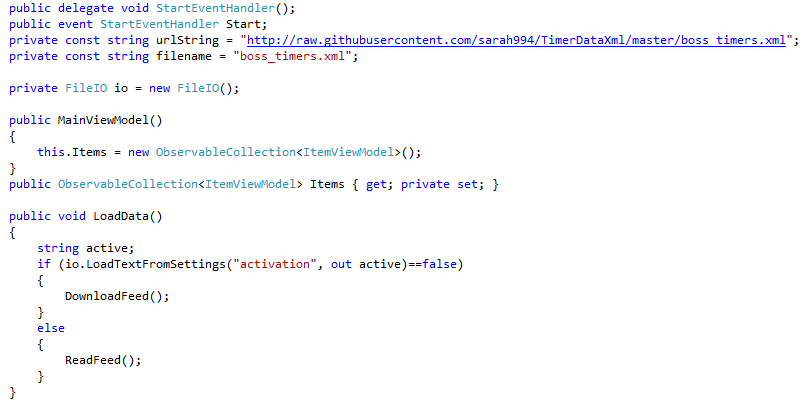
Deze klasse bevat de achterliggende code van de grafische interface. De eerste methode is OnNavigatedTo. Deze start het proces waarbij de data wordt opgehaald en gebonden. Er staat een delete bij om te voorkomen dat de pagina getoond wordt voordat alle data geladen is, anders blijven de textBlocks leeg. De methode ViewModel\_Start() wordt dan geladen en koppelt het juiste pivotItem me de juiste methode. Er is ook een dispatchTimer die in lambda vorm is geschreven. Deze herlaadt om de seconden de pivots zodat ik een seconden countdown kan tonen in de pivots en de waarden om de 15minuten verandert.



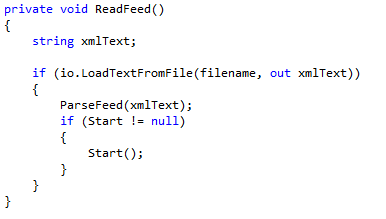
## MainViewModel.cs

Deze klasse bevat het grootste deel van al mijn code. Eerst worden properties geinitialiseerd. Zoals de delegate Start. De URL van het xml bestand wordt opgegeven en een naam om het in een isolated storage op te slaan. De klasse FileIO bevat alle code voor de isolated storage en wordt ook al aangemaakt. Als laatste heb je de observableCollection Items die aangemaakt wordt en alle ingelezen data zal meekrijgen. ObservableCollection is een klasse voor dynamische data in op te slaan. Er wordt een melding gegeven als er iets verandert.

De eerste methode die zal worden aangeroepen door de MainPage is LoadData(). Het checkt als dit de eerste keer is dat informatie moet worden opgehaald. Dit wordt met behulp van de isolated storage gedaan waar “activation” een false teruggeeft als de code nog nooit eerder gebruikt was.



Indien de code eerder geactiveerd werd, wordt het xml bestand uitgelezen uit de isolated storage. De lijst wordt doorgestuurd naar ParseFeed() en de delegate wordt heropgeroepen omdat het laden van het xml-bestand klaar is.

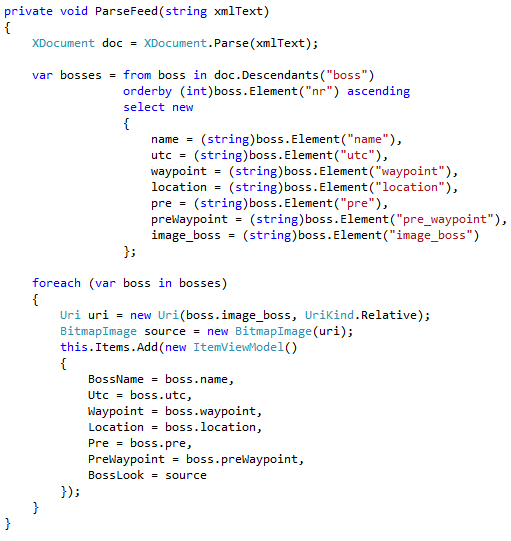


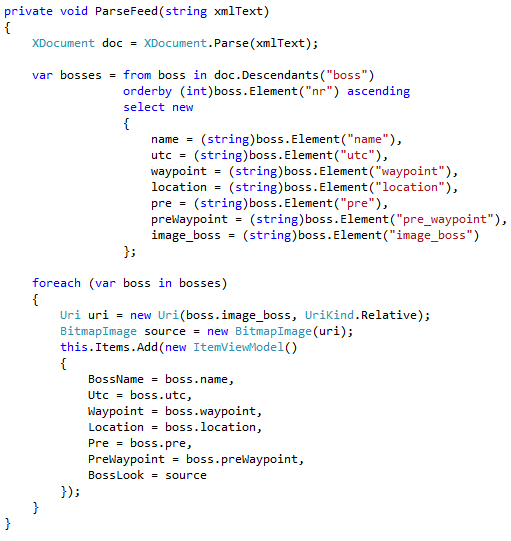
Als de code nog nooit eerder werd ingeladen, zal dit door DownloadFeed() van het internet gehaald worden. WebClient is een klasse voor het versturen of ontvangen van data van uri. Een async leest de uri uit en een delegate roept een tweede methode aan. Deze leest de xml in een string in en stopt als deze klaar is. Deze string wordt naar ParseFeed gestuurd en opgeslagen in de isolated storage. Ook “activation” waarmee LoadData test of er al eerder code is geweest wordt nu aangemaakt en op true gezet. De startEventHandler delegate wordt weer aangeroepen nu dat de xml is uitgelezen.



ParseFeed ontvangt het xml bestand en leest dit via XDocument uit. Ik heb een nummering toegevoegd aan mijn xml om problemen te vermijden bij het inladen en daarop zal de data dan ook geordend zijn. Alle informatie wordt met een foreach in Items opgeslagen met de benaming van de van de ItemViewModel klasse.

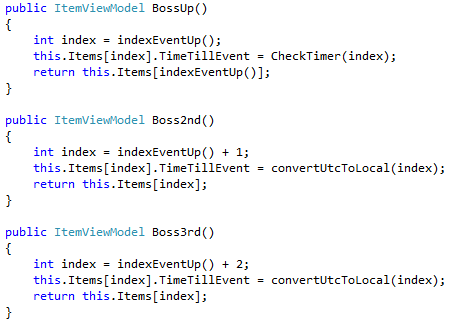
Om de afbeeldingen te databinden had ik wat problemen, maar de oplossing was simpel. 2 regeltjes code die de string omzetten naar een uri en dan naar een bitmapImage source. Zo wist het programma dat dit een afbeeldingspad was.



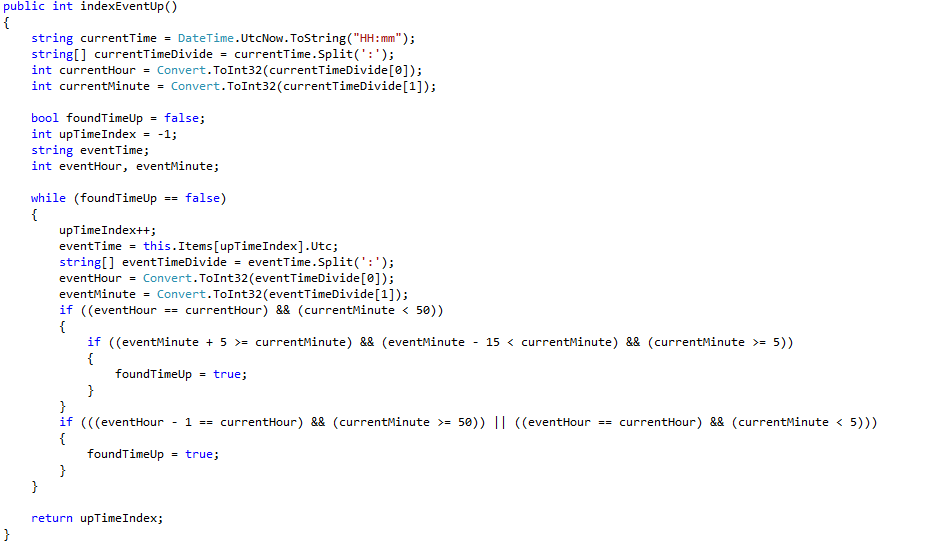


De volgende 3 methoden worden aan de pivots gekoppeld en moeten dus de juiste data meegeven op dat moment. De index wordt opgevraagd en gebruikt om specifieke data uit de Items te halen. Maar eerst wordt de tijd tot de event voor elke methoden aangepast. De BossUp() moet een countdown tonen tot dat event plaatsvindt en daarna de event tijd in lokale tijd tonen. Boss2nd() en Boss3rd() moeten constant de lokale tijd van het event tonen. Alle methoden worden door een zelf geschreven methode geconverteerd.

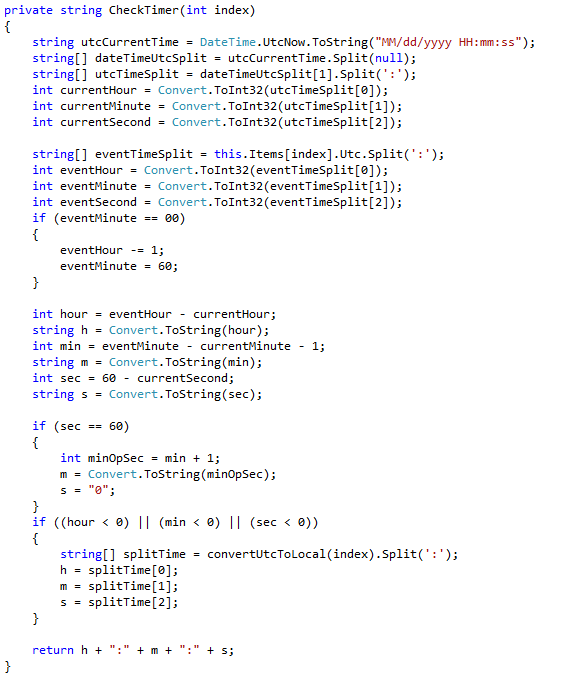
Als de data eindelijke compleet is wordt deze doorgestuurd als ItemViewModel.

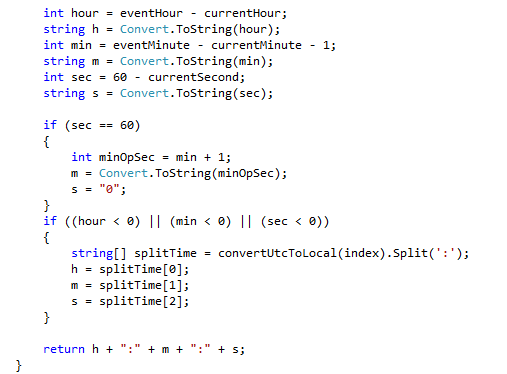


De volgende code leest de huidige UTC waarde in en doorloopt de lijst totdat het de event tijd vind die het dichtste bij de huidige tijd aansluit. De boolean foundTimeUp wordt dan op true gezet, de functie stopt met zoeken en stuurt het getal waarop het stopte terug als index.

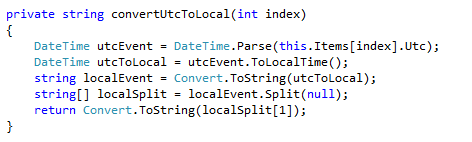


CheckTimer() wordt gebruikt door BossUp en dient zowel als countdown als UTC naar lokale tijd convertor. De huidige UTC tijd in wordt opgeslagen in afgezonderde integers voor uur, minuut en second. Deze worden vergeleken met het uur, minuut en second van de event time van de index die meegestuurd werd. Het verschil tussen deze twee tijden wordt meegestuurd in een string. Als uur, minuut en second overal als verschil 0 hebben, zal de code event tijd in lokale tijd meesturen in een string. Deze wordt net zoals door Boss2nd() en Boss3rd() uit een andere methode gehaald.



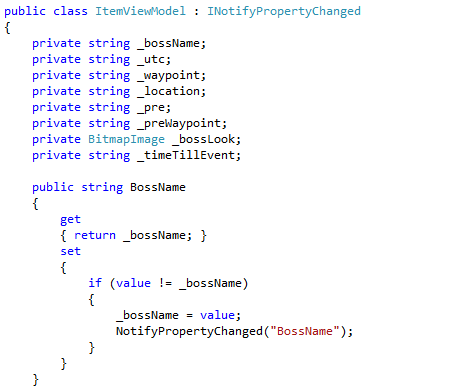


Converteert de UTC tijd naar de lokale tijd van de computer. De UTC wordt geparsed naar DateTime zodat ik ToLocalTime() kan aanroepen en de conversie automatisch gebeurd.

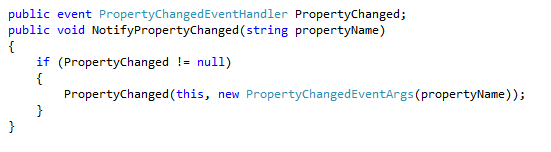


## ItemViewModel.cs

ItemViewModel is een klasse die gebruikt wordt door ObservableCollection om voor 1 index meerdere data te onthouden. Wanneer er iets verandert zal er een melding gegeven worden, dit gebeurd via INotifyPropertyChanged. Voor elke waarde wordt een variabele gedeclareerd en in een get-set met dezelfde naam gebruikt. Onderstaande BossName is de manier waarop al deze get-sets worden gebruikt.

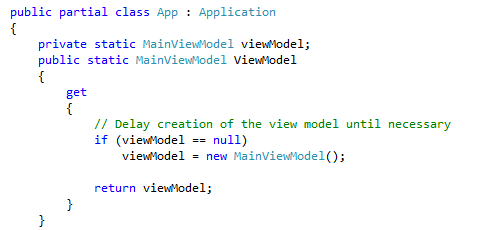


Wanneer value geset wordt zal de methode NotifyPropertyChanged worden geroepen.



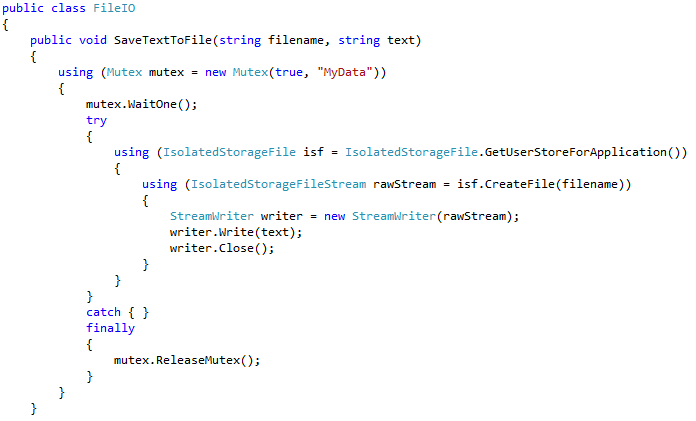
## App.xaml.cs

Is de globale klasse waar je de globale variabelen kunt opslaan. De MainViewModel wordt hier in een get-methode gestopt met de naam ViewModel. Zo kan de MainPage aan de informatie binnen de MainViewModel.



## FileIO.cs

Een klasse die we in de les hebben aangemaakt en enorm handig is om compact gegevens op te slaan in de localStorage en uit te lezen.







# Afbeeldingen

## Achtergrond afbeelding

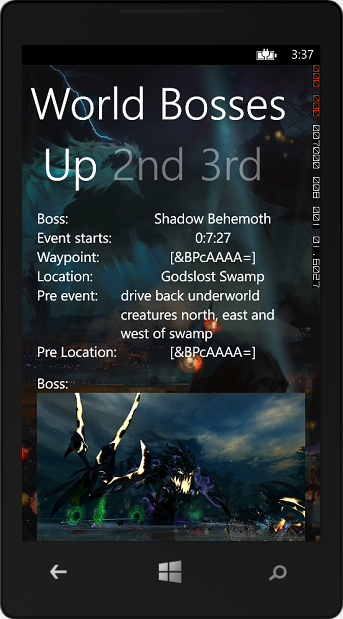
De achtergrond afbeelding die over heel de applicatie wordt gebruikt.



## Screenshots

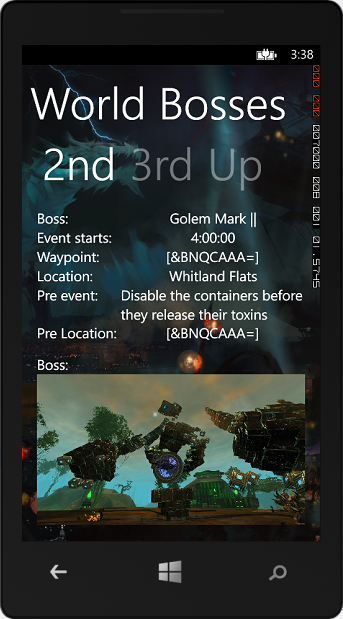
### Up pivotItem

Een screenshot van de applicatie wanneer de countdown aan het aftellen is op de Up pivotItem.



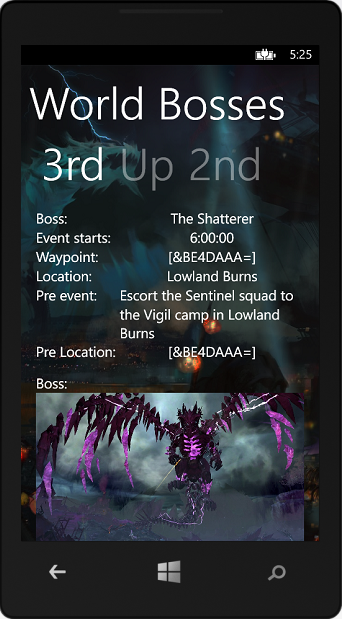
### 2nd pivotItem

Een screenshot van de applicatie op de 2nd pivotItem.



### 3rd pivotItem

Een screenshot van de applicatie op de 3rd pivotItem.



# Slot

Doorheen het project heb ik geleerd dat ik sneller uitleg moet vragen en een beter zicht moet hebben over wat mijn applicatie moet kunnen voordat ik eraan begin. Ik heb bijvoorbeeld een bijzondere lange tijd problemen gehad met het koppelen van afbeeldingspad via ItemViewModel(). Via het internet heb ik gezocht naar een oplossing, maar dit duurde enige tijd. Achteraf gezien had ik dit kleine detail beter kunnen vragen in de les, want de oplossing bestond uit 2 simpele regels. Er zijn zo meerdere problemen geweest die ik ofwel via internet of tijdens de les heb opgelost. Uiteindelijk ben ik tevreden met wat ik nu heb bereikt.