

Lab Programmeertechnieken

Rekenapplicatie: Math Game

Timon Claerhout

Docent: L. Espeel

Academiejaar: 2022 - 2023

Inhoudsopgave

1	INLEIDING	3
2	OPBOUW APPLICATIE.....	4
2.1	DATA LAAG	4
2.2	BUSINESS LAAG.....	6
2.2.1	<i>Player klasse</i>	<i>7</i>
2.2.2	<i>MathChallenge klasse</i>	<i>8</i>
2.2.3	<i>Game klasse.....</i>	<i>9</i>
2.3	PRESENTATIE LAAG	10
2.3.1	<i>Login venster.....</i>	<i>11</i>
2.3.1.1	Moeilijkheidsgraad makkelijk.....	12
2.3.1.2	Moeilijkheidsgraad normaal.....	12
2.3.1.3	Moeilijkheidsgraad moeilijk.....	12
2.3.2	<i>Hoofd venster</i>	<i>13</i>
2.3.2.1	Spelverloop moeilijkheidsgraad makkelijk	14
2.3.2.2	Spelverloop moeilijkheidsgraad normaal & moeilijk	16
2.3.3	<i>Eind venster.....</i>	<i>18</i>
2.3.3.1	Speleinde moeilijkheidsgraad makkelijk	18
2.3.3.2	Speleinde moeilijkheidsgraad normaal & moeilijk	19
2.3.4	<i>Scorebord venster</i>	<i>21</i>
3	UNIT TEST	22
4	BIJLAGEN & BRONNEN.....	23

1 Inleiding

Het vak programmeertechnieken bestaat zowel uit een deel theorie als een deel labo opdrachten. Als opdracht voor het labo programmeertechnieken is het de bedoeling om een applicatie te ontwikkelen geprogrammeerd in C, C++ en C#. Het onderwerp van deze applicatie is vrij te kiezen, althans is het wel de bedoeling om zoveel mogelijk aangeleerde programmeertechnieken binnen deze applicatie te verwerken.

Ik heb gekozen om een wiskundige spelapplicatie te maken namelijk “Math Game”.

Het doel van de Math Game is om zowel kinderen als volwassenen op een toffe manier hun wiskunde kennis op te frissen en te onderhouden, via een overzichtelijke en robuuste applicatie. Deze app zal 10 keer 2 willekeurige getallen weergeven met een willekeurige operator die de gebruiker zo snel mogelijk moet oplossen. Om het spel uitdagend te maken voor volwassenen maken we gebruik van verschillende moeilijkheidsgraden.

Ook moet er wat spanning in het spel zitten, daardoor is er een scorebord geïmplementeerd. Dit scoreboard moet steeds geordend worden met de spelers die zowel de hoogste score als de snelste tijd hebben om de rekensommen op te lossen. Om het spel interactief te maken gaan er ook geluidseffecten, knoppen die veranderen van kleur, hover-effecten en zoveel meer voorzien zijn.

Dit alles moet op een object georiënteerde manier geschreven worden in Visual Studio Code 2022 via de recentste versie. Met zo min mogelijke code duplicatie om zo de performantie zo hoog mogelijk te houden. Hoe ik dit alles heb aangepakt zal beschreven worden op onderstaande pagina's maar ook in de source code die volledig gedocumenteerd is met behulp van Doxygen-commentaar.

2 Opbouw applicatie

Om het volledige project overzichtelijk te houden en gemakkelijk delen van de applicatie te kunnen hergebruiken of aanpassen is het een goede gewoonte om de applicatie op te splitsen in verschillende lagen en deelprojecten namelijk: **presentatie-, business- en data laag**.

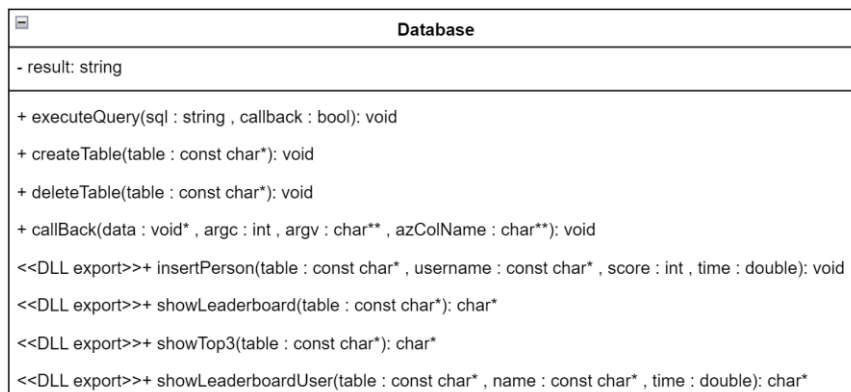
- De data laag wordt gebruikt voor het opslaan van data en het aanspreken van de database met specifieke queries.
- In de business laag programmeren we de logica en functionaliteit van de applicatie met verschillende klassen. Het resultaat van deze logica zal weergegeven worden via de presentatie laag.
- In de presentatie laag programmeren we de GUI of graphical user interface. In deze laag bepalen we hoe de applicatie er visueel zal uit zien en welke functionaliteit deze zal hebben zoals knoppen en tekstvelden. Dit is de enige laag dat de gebruiker ziet die het spel speelt.

2.1 Data laag

Deze laag zorgt voor de connectie met de database die verwezenlijkt wordt via SQLite3 geschreven in ISO C++20 Standard. Ik heb gekozen voor SQLite3 omdat dit iets nieuws was voor mij om toe te passen in een applicatie, maar ook door onder andere volgende voordelen:

- Geen installatie vereist
- Kan op iedere operating system gebruikt worden
- Vraagt veel minder geheugen dan bv. SQL database
- Kan snel queries uitvoeren en data uit de databank ophalen
- Alle gegevens worden lokaal opgeslagen waardoor geen internet vereist is

Onderstaande UML diagram geeft weer welke functies en variabelen gebruikt zijn om deze databank te verwezenlijken:



De executeQuery functie wordt door alle andere functies wordt aangesproken doordat enkel deze functie een query rechtstreeks naar de database stuurt. Wanneer een functie zijn query wilt doorsturen geeft hij deze string mee als 1^{ste} parameter van de executeQuery functie. Via de 2^{de} parameter wordt een boolean meegeven indien een callback vereist is:

- Wanneer deze **boolean true** is, zal de callback functie die standaard geïntegreerd zit in de SQLite3 library aangesproken worden. Dit zal gebruikt worden wanneer de doorgegeven query een resultaat terug verwacht wordt. Hiervoor gaat de inhoud van het desbetreffend resultaat toegekend worden in de private string result. Alle functies die een char* teruggeven zullen de callback functie gebruiken.
- Wanneer deze **boolean false** is, zal de callback functie niet gebruikt worden omdat er geen resultaat verwacht wordt door de query die werd uitgevoerd in de database. Dit is het geval bij alle functies die void teruggeven.

De functies createTable en deleteTable worden gebruikt voor de allereerste keer wanneer de databank moet worden opgezet. Deze is dus niet nodig voor de gebruiker maar wel voor de programmeur, hetzelfde met de callback en executeQuery functie. Daardoor worden deze functies ook niet geëxporteerd als DLL functie en de andere (die wel nodig zijn voor de gebruiker in het spel) wel.

Bij het opzetten van de databank heb ik "Database.db" aangemaakt met daarin 2 verschillende tables doordat er 2 verschillende moeilijkheidsgraden zijn. Deze tables worden "Normal_Leaderboard" en "Hard_Leaderboard" genoemd. Deze table namen zullen worden gebruikt voor de DLL functies die geëxporteerd worden zodat de juiste data wordt weergegeven bij de corresponderende graad.

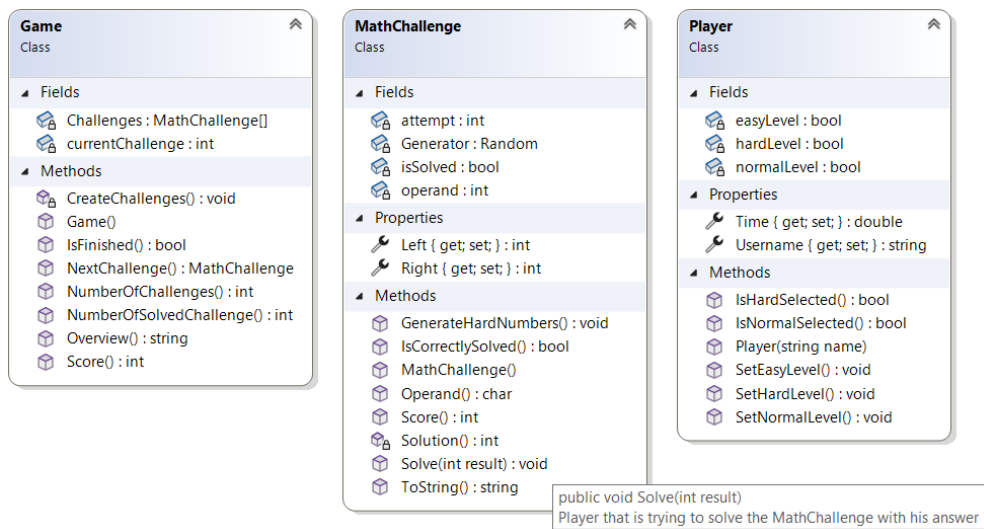
Via insertPerson wordt een nieuwe speler in het scorebord toegevoegd in de juiste table. Via de 1^{ste} parameter wordt de geschikte table meegeven, daarna de gebruikersnaam gevolgd met zijn bijhorende score en ten slotte de tijd dat de speler nodig had om de rekensommen op te lossen.

ShowLeaderBoard zal het gehele leaderboard weergeven van de specifieke table die meegeven wordt als 1^{ste} parameter. Dit door allereerst de data te ordenen door de spelers die de hoogste score met de snelste tijd hebben eerst te plaatsen. Via de callback functie zal dit resultaat weergegeven worden op de private string result, waarbij deze string zal teruggeven worden na een cast te doen naar char* zodat in de managed C# applicatie deze kan terug casten naar string via IntPtr klasse. Hetzelfde wordt gedaan voor de showTop3 en showLeaderBoardUser functie. De showLeaderBoardUser functie zal de plaats en geleverde prestatie van de speler weergeven op de corresponderende table dat hij heeft gespeeld. Dit wordt gedaan door te zoeken op de naam van speler en de bijhorende tijd. Aangezien de tijd op 0.001 seconden nauwkeurig is en (bijna) alle verschillende spelers een andere naam hebben is dit een geschikte manier om hierop te filteren.

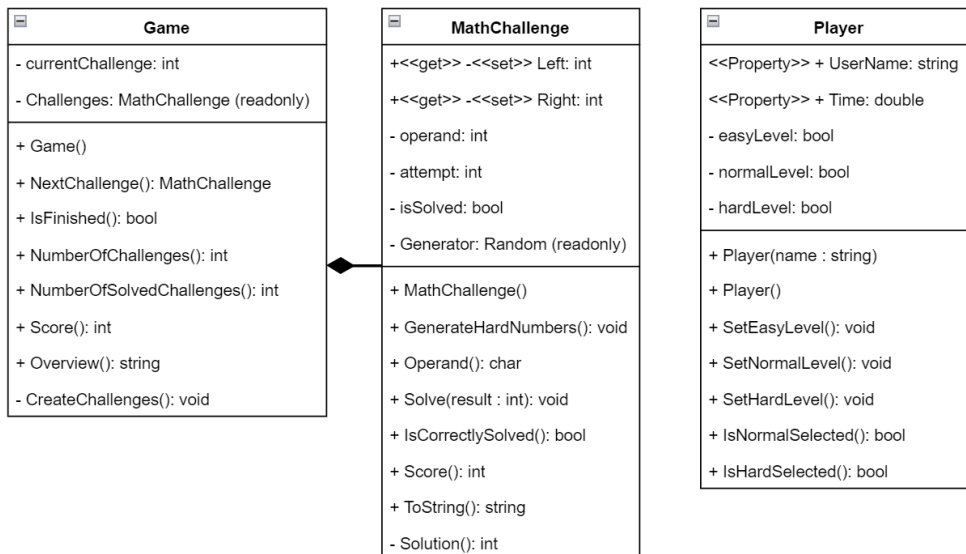
2.2 Business laag

Deze laag bestaat uit 3 verschillende klassen: Game, MathChallenge en Player. Zoals eerder besproken gaan deze klassen aangesproken worden door de presentatie laag, dit omdat het resultaat van deze klassen de spel logica bevat. Deze 3 klassen worden op de volgende pagina in detail besproken. Met behulp van een zelf geschreven UML diagram gaan ook alle return en parameter types aan bod komen om de implementatie te verduidelijken. Uiteraard is er ook een globaal UML diagram gemaakt dat is weergegeven in “BusinessLayerUML.cd”. Dit zal een overzicht geven over alle gebruikte variabelen en methodes van de 3 klassen. Wanneer je dit bestand opent en je muis over een item houdt, krijg je alsook een bondige uitleg over wat de functie inhoudt (dit door Doxygen).

Zie UML diagram hierbeneden:



Klassenrelatie:



2.2.1 Player klasse

De Player klasse wordt gebruikt om alle gegevens bij te houden die de gebruiker heeft gekozen aan het begin van de Math Game zoals zijn gebruikersnaam en moeilijkheidsgraad. Maar ook op het einde van het spel wordt deze klasse gebruikt om de geleverde prestaties op te slaan zoals de tijd dat de speler nodig had om de rekensommen op te lossen. Hieronder wordt de implementatie besproken.

Bij het starten van een spel wordt de default constructor Player gebruikt, deze zal de klasse initialiseren en de gebruikersnaam bijhouden via een publieke property functie UserName. Wanneer de gebruiker het spel heeft gespeeld en opnieuw wilt spelen dan zal de non-default constructor Player(string name) gebruikt worden. Dit omdat de gebruikersnaam al eerder is ingegeven en deze naam meteen opnieuw gebruikt zal worden. Zodat de gebruiker niet telkens eenzelfde naam moet ingeven. Ten slotte kiest de gebruiker zelf in welke moeilijkheidsgraad wordt gespeeld. Dit wordt verwezenlijkt door de 3 private booleans die gebruikt worden via de corresponderende publieke Set functies. Hier is er gekozen om niet met Properties te werken en dit heeft zijn reden.

Wanneer de gebruiker graad normaal aanklikt wordt SetNormalLevel methode aangesproken, deze zal de boolean normalLevel op true zetten. Maar deze moet de andere 2 booleans op false zetten omdat er anders meerdere booleans op true komen waardoor de juiste moeilijkheidsgraad niet meer achterhaald kan worden. Ten slotte is er nog een publieke property Time van het type double, die de tijd dat de gebruiker nodig had om de rekensommen op te lossen bijhoudt.

Hoe dat de verschillende moeilijkheidsgraden worden geïmplementeerd zal verder besproken worden in het [login venster](#).

Player
<<Property>> + UserName: string
<<Property>> + Time: double
- easyLevel: bool
- normalLevel: bool
- hardLevel: bool
+ Player(name : string)
+ Player()
+ SetEasyLevel(): void
+ SetNormalLevel(): void
+ SetHardLevel(): void
+ IsNormalSelected(): bool
+ IsHardSelected(): bool

2.2.2 MathChallenge klasse

De MathChallenge klasse wordt gebruikt om de verschillende rekensommen en operatoren te generen via de moeilijkheidsgraad. Maar bekijkt ook als de gebruiker deze correct heeft opgelost en een punt scoort of niet. Hoe dit in zijn werk gaat wordt besproken via het gedetailleerde UML diagram hieronder.

De default constructor MathChallenge wordt meteen uitgevoerd bij het aanmaken van deze klasse. Deze begint met de .NET klasse Random aan te roepen om een willekeurige operator te generen. Dit door in de private operand variabele een random getal toe te kennen tussen 1-4 (omdat er 4 verschillende operatoren mogelijk zijn). Deze random waarde wordt in een switch statement gebruikt waarbij een universeel formaat wordt gebruikt binnenin deze klasse. Dit formaat gaat als het volgt:

- case 1 = '+' operator -> random getallen tussen 0 - 20
- case 2 = '-' operator -> random getallen tussen 0 - 20
- case 3 = '*' operator -> random getallen tussen 0 - 20 en 0 - 10
- case 4 = '/' operator -> random getal tussen 4-25 en zoeken voor een deelbaar ander nummer via een do while statement

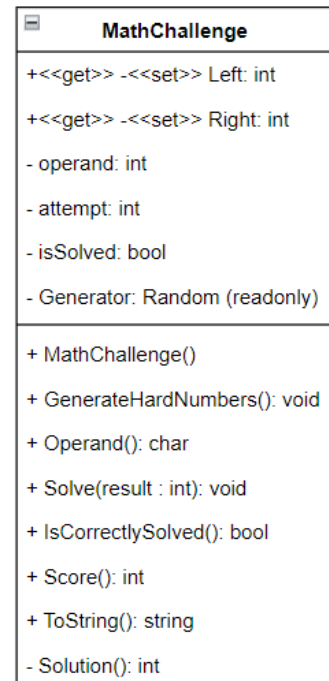
Deze gegenereerde getallen worden toegekend aan de corresponderende Properties Left en Right. Stel dat de gebruiker op moeilijkheidsgraad moeilijk speelt gaat GenerateHardNumbers gebruikt worden om de constructor waarden te overschrijven in Left en Right. Deze zullen volgende getalwaarden gebruiken:

- case 1 en 2 getallen gebruiken tussen 0-100
- case 3 het linkse getal tussen 20-40 en de rechtse tussen 3-20, merk op nu zal er ook geen *0 rekensom gegeven worden
- case 4 wordt wederom een deelbaar getal gezocht maar deze mag niet meer deelbaar zijn door zichzelf of door 1, ook gaat er een willekeurig getal tussen 4-100 (Left) gedeeld worden tussen een getal tussen 2-50 (Right)

De Operand functie zal via het besproken switch formaat de juiste operator teruggeven via het char datatype aangezien er maar 1 karakter vereist is.

De Solution functie zal via eenzelfde switch formaat het juiste antwoord van de rekensom teruggeven als integer door de properties Left en Right te combineren met de bijhorende operator.

De functie Solve(int result) zal als parameter het resultaat dat de gebruiker heeft ingetypt om de rekensom op te lossen bijhouden. Hij zal de waarde van dit



resultaat toekennen aan de private integer attempt en de private boolean IsSolved op true zetten.

De IsCorrectlySolved functie zal de attempt variabele gebruiken om dit te vergelijken met de Solution integer en aan de hand daarvan een boolean teruggeven.

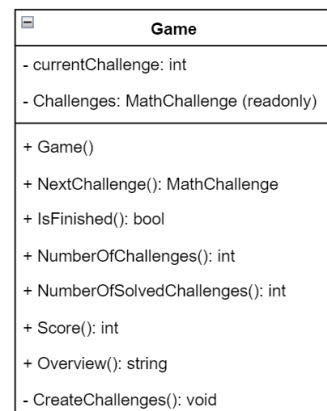
De Score functie zal aan de hand van de IsCorrectlySolved functie bij een correct antwoord een '1' teruggeven of bij een fout antwoord '0'.

Ten slotte hebben we de ToString functie die overschreden is doordat dit een basis functie is. Deze zal een string teruggeven met een overzicht van de gevraagde rekensommen en antwoorden van de gebruiker. Bij een correct antwoord wordt een "[V]" weergegeven, bij een fout "[X]" met het correcte antwoord er naast. Wanneer er geen oplossing is gegeven door de gebruiker zal een "?" weergegeven worden. Dit wordt gecheckt via de IsSolved boolean.

2.2.3 Game klasse

De Game klasse wordt gebruikt om met behulp van de MathChallenge 10 verschillende rekensommen te genereren. Via deze rekensommen zal hij via allerlei functies bijhouden in welke rekensom de gebruiker bezig, wanneer de volgende moet gegeven worden, wanneer de gebruiker klaar is met de rekensommen uit te voeren, etc. De implementatie wordt hieronder gegeven via het UML diagram.

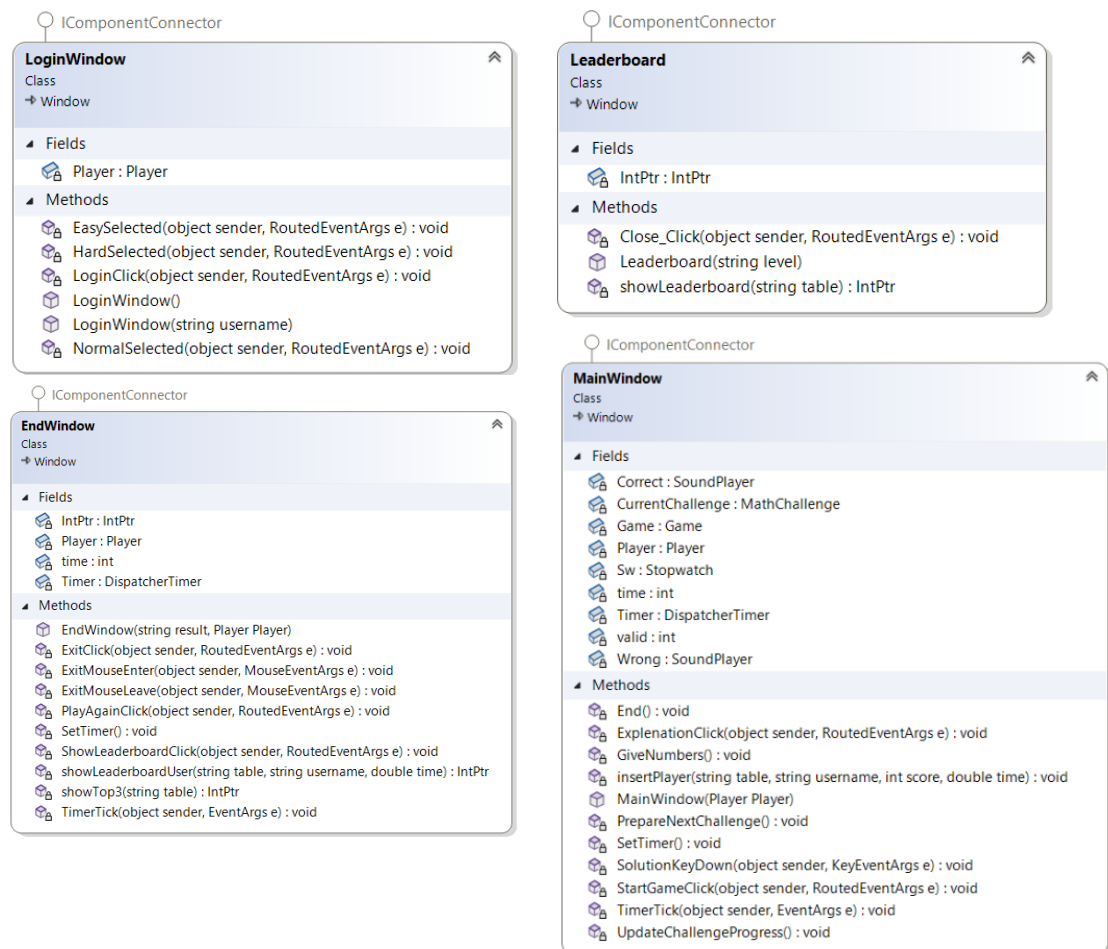
De default constructor Game wordt allereerst aangesproken bij het aanmaken van deze klasse. Hierin wordt de private functie CreateChallenges gebruikt. Deze zal via de private Challenges array van type MathChallenge, 10 nieuwe elementen (MathChallenges) toekennen hierin. De private variabele currentChallenge zal hierbij de index bijhouden in welke MathChallenge van de Challenges array de gebruiker bezig is. Door gebruik te maken van NextChallenge functie zal de nieuwe MathChallenge in de array teruggegeven worden en zal currentChallenge met 1 verhoogd worden. Uiteraard is er een validatie voorzien via de IsFinished functie. Deze kijkt als de gebruiker niet op zijn laatste MathChallenge zit, als dit het geval is kan de currentChallenge niet meer verhoogd worden in de array. De functies NumberOfChallenges (aantal MathChallenges in de array) en NumberOfSolvedChallenges (aantal opgeloste MathChallenges in de array) worden vergeleken met elkaar als deze gelijk zijn dan is de boolean true, zo niet is deze false.



Ten slotte zal de Overview functie een overzicht geven van de gevraagde rekensommen. Dit wordt verwezenlijkt door via een foreach loop de Challenges array te overlopen en naar een string object te casten. Hierdoor zal de overgeschreven ToString methode aangesproken worden van de MathChallenge klasse waardoor de correcte rekensommen met bijhorende oplossing gegeven worden. Daaronder wordt de totale score van de gebruiker weergegeven via de functie Score(). Deze haalt de totale score op door wederom via een foreach loop de array te overlopen en de Score() functie van MathChallenge te gebruiken. Deze wordt overlopen bij alle 10 indexen en worden steeds met elkaar opgeteld.

2.3 Presentatie laag

De presentatie laag is geschreven via WPF. Deze laag bestaat uit 4 vensters waarin de gebruiker zal spelen. Deze 4 vensters worden op de volgende pagina in detail besproken. Hieronder is het globaal UML diagram weergegeven uit "PresentationLayerUML.cd", wat een overzicht geeft over alle gebruikte variabelen en methodes van de vensters. Wanneer je dit bestand opent en je muis over een item houdt, krijg je alsook een bondige uitleg over wat de functie is (dit door Doxygen). Zie diagram hierbeneden:



Tevens zijn alle vensters volledig “responsive” gemaakt. M.a.w. de applicatie past zich aan met het scherm van de gebruiker, als dit op een pc, tablet of smartphone is de mooie lay-out zal behouden blijven. Dit is verwezenlijkt door gebruik te maken van “Grid” met bijhorende “Grid.RowDefinitions” en “Grid.ColumnDefinitions”.

Wanneer een bepaald scherm groter/kleiner is kan de leesbaarheid van de letters niet goed zijn aangezien het lettertype dezelfde blijft. Dit kan opgelost worden door “ViewBox” toe te voegen bij de “Grid”. Dit zorgt er voor dat ook het lettertype “responsive” wordt.

Hierdoor wordt er een maximaal visueel comfort aangeboden aan de gebruiker.

2.3.1 Login venster

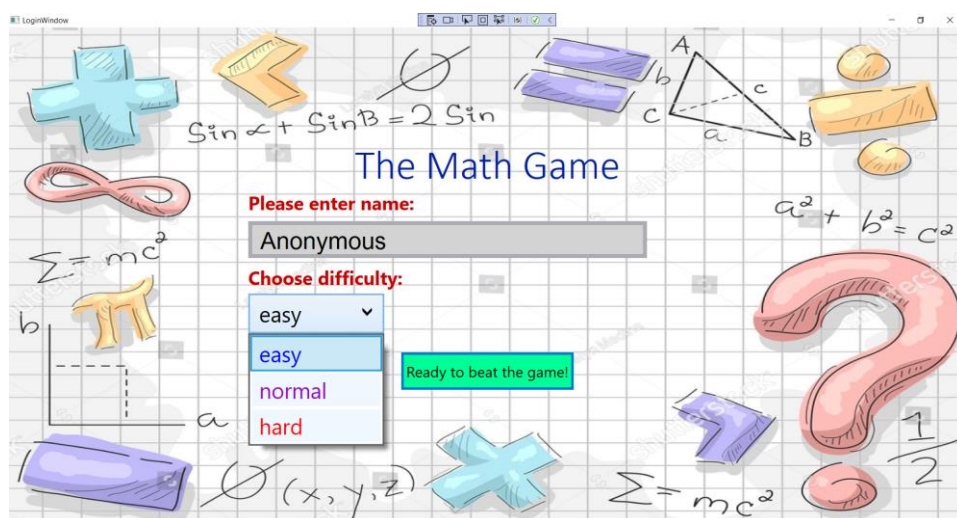
In het login venster wordt de gebruikersnaam en de moeilijkheidsgraad gevraagd.

Om de applicatie robuust te maken is het aantal karakters dat de gebruiker kan intypen gelimiteerd op 15. Omdat anders de naam enorm lang kan worden wat kan leiden tot niet gewenste resultaten of fouten. Dit wordt voorzien door in de gebruikersnaam “TextBox” de parameter “MaxLength=“15” ” toe te voegen.

Ook is er een validatie voorzien als de “TextBox” niet leeggelaten is. Indien dit het geval is komt er een pop-up (“MessageBox”) tevoorschijn die vraagt aan de gebruiker om een naam in te typen. Standaard is de naam “Anonymous” voorzien als de gebruiker liever zijn/haar eigen naam niet wilt opgeven.

Via een “ComboBox” wordt de moeilijkheidsgraad geselecteerd. De gebruiker heeft de keuze uit 3 verschillende graden: makkelijk, normaal en moeilijk.

Hieronder is de lay-out van het bekomen login venster met implementatie:



De waarden die de gebruiker intypt bij het loginvenster bevat de achterliggend de logica die gebruikt wordt door onderstaande UML diagram:

De code start bij de default constructor LoginWindow waarbij een nieuwe Player klasse wordt aangemaakt met de ingegeven gebruikersnaam. De 3 selectoren zijn verbonden met de “ComboBox” die afhankelijk daarvan de juiste moeilijkheidsgraad zullen selecteren in de Player klasse. Ten slotte wordt de “LoginClick” aangesproken wanneer de gebruiker op de “Ready to beat the game!” knop drukt als hij het spel wilt starten. In deze methode zal het hoofd venster aangesproken worden met de gehele Player klasse als argument, zodat de private speler instellingen meegegeven wordt. Dit venster zal in detail verder besproken worden in [kop 2.1.2 Hoofd venster](#).



De 3 moeilijkheidsgraden hebben uiteraard elk hun eigen implementaties. Deze zullen hieronder verder toegelicht worden.

2.3.1.1 Moeilijkheidsgraad makkelijk

Deze graad is bedoelt voor kinderen die leren rekenen en dit willen oefenen. Hiervoor krijgen ze zoveel tijd dat ze zelf nodig hebben om de wiskundesommen op te lossen met kleinere getallen. Er is dan ook geen scoreboard voorzien voor deze graad aangezien iedere speler op zijn eigen tempo speelt.

2.3.1.2 Moeilijkheidsgraad normaal

Deze graad is bedoelt voor jongvolwassenen doordat je hier niet op je eigen tempo speelt. Je krijgt namelijk 5 seconden de tijd om een rekensom op te lossen anders krijg je geen punt en wordt de volgende rekensom weergegeven. De tijd dat de gebruiker nodig had om de rekensommen op te lossen wordt ook bijgehouden. Dit zodat zowel de score en de tijd wordt meegegeven aan de database die het scorebord zal voorzien. Deze zal ook constant geordend worden met de spelers die de hoogste score en snelste tijd hebben bovenaan.

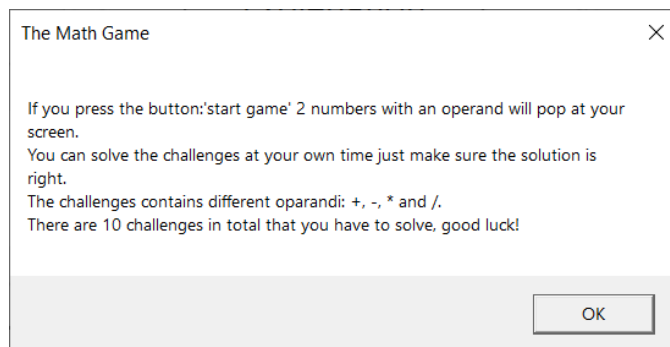
2.3.1.3 Moeilijkheidsgraad moeilijk

Deze graad is bedoelt voor volwassenen doordat je hier niet op je eigen tempo speelt. Je krijgt namelijk 5 seconden de tijd om een rekensom op te lossen anders krijg je geen punt en wordt de volgende rekensom weergegeven. Deze rekensom bevat uiteraard nog grotere, moeilijkere getallen dan in graad normaal. De tijd dat de gebruiker nodig had om de rekensommen op te lossen wordt ook bijgehouden.

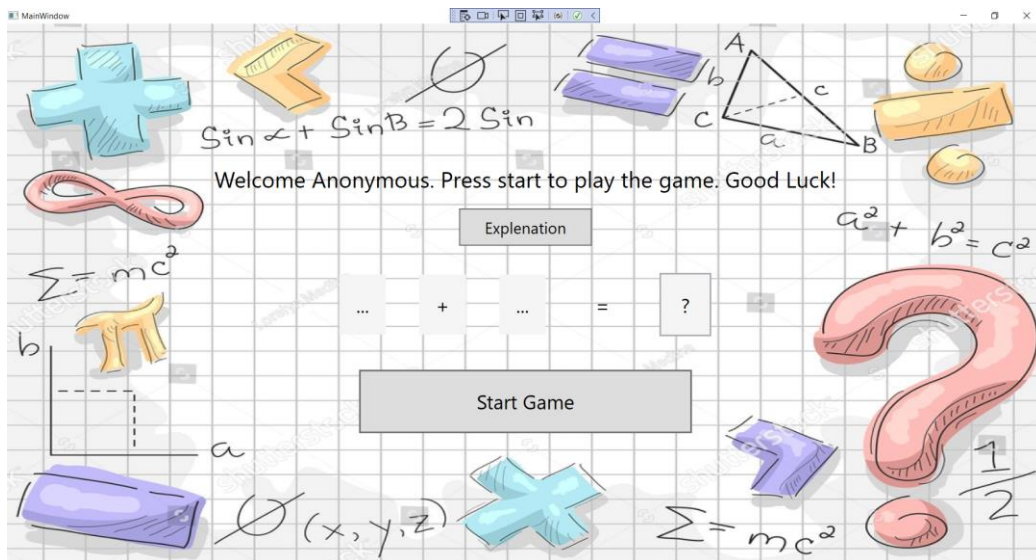
Dit zodat zowel de score en de tijd wordt meegegeven aan de database die het scorebord zal voorzien. Deze zal ook constant geordend worden met de spelers die de hoogste score en snelste tijd hebben bovenaan.

2.3.2 Hoofd venster

Het hoofd venster bevat de effectieve spel logica waarin de gebruiker allereerst de keuze krijgt om uitleg te krijgen bij het spelen van het spel. Hiervoor wordt een pop-up ("MessageBox") gebruikt die de gepaste uitleg voorziet afhankelijk van de moeilijkheidsgraad. Hieronder is het pop-up bericht weergegeven bij moeilijkheidsgraad makkelijk:



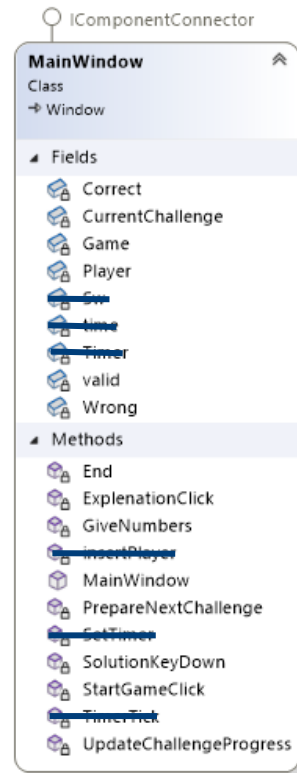
Pas bij het drukken op de "Start game" knop worden de rekensommen gegeven. De lay-out van dit beginvenster ziet er als volgt uit:



2.3.2.1 Spelverloop moeilijkheidsgraad makkelijk

In deze graad worden niet alle functies gebruikt aangezien er geen scorebord of tijden worden bijgehouden. Deze zijn dan ook doorstreept op het UML diagram:

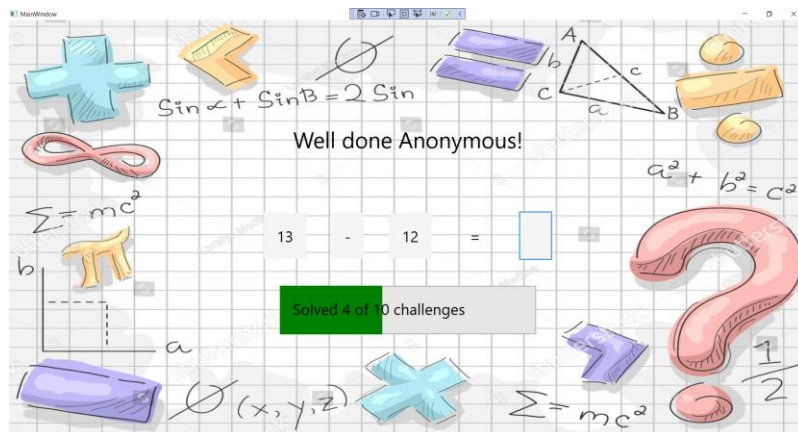
De code start uiteraard in de default constructor MainWindow waarbij de 2 geluidseffecten eerst asynchroon worden ingeladen via de .NET klasse SoundPlayer. Deze worden gebruikt als de gebruiker een juist/fout antwoord geeft. Daarna kan de gebruiker kiezen zoals eerder uitgelegd om uitleg te krijgen, hiervoor wordt de ExplanenationClick methode gebruikt. Wanneer de gebruiker het spel wilt starten zal de StartGameClick methode opgeropen worden doordat er op de corresponderende drukknop is geklikt. Deze zal de Game klasse oproepen die 10 verschillende soorten rekensommen bevat. Daarna zal de drukknop verdwijnen en veranderen door een voortgangsbalk. Deze geeft weer aan welke rekensom van de 10 de gebruiker zit. Ook zal het kleur veranderen in groen bij een juist antwoord en rood bij een fout. Dit wordt verwezenlijkt door de UpdateChallengeProgress methode. De methode PrepareNextChallenge zal deze aanspreken doordat de vooruitgangsbalk telkens moet worden geüpdatet en de nieuwe rekensom gegeven, dit door ook de GiveNumers methode aan te roepen. Daardoor gaan de nummers van de corresponderende rekensom weergegeven worden in het hoofd venster.



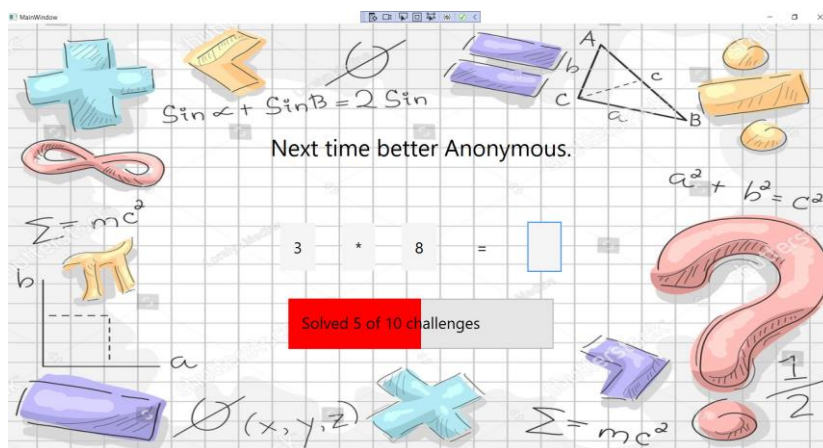
Wanneer de gebruiker de rekensom wilt oplossen is er ook een validatie voorzien op het ingeven van het antwoord via de methode SolutionKeyDown. Dit door gebruik te maken van exceptie afhandeling via try&catch , hierdoor wordt er gekeken als de gebruiker weldegelijk een cijfer heeft ingegeven en geen letters, leestekens of andere ongewenste tekens. Wanneer dit het geval is moet het antwoorden tekst vak leeggemaakt worden en de gebruiker de kans geven om een antwoord te geven met de gepaste cijfers.

Op de volgende pagina is te zien hoe de lay-out weergegeven wordt.

De lay-out van het speelscherm bij een correct antwoord ziet er als volgt uit:



De lay-out van het speelscherm bij een fout antwoord ziet er als volgt uit:



Wanneer de gebruiker ten einde komt van de 10 verschillende rekensommen wordt de methode End aangesproken. Dit zal het eind venster aanspreken die een overzicht meegeeft met de gevraagde rekensommen, corresponderend antwoord van de gebruiker en de score die de gebruiker behaald heeft. Het eind venster wordt verder besproken bij [kop 2.1.3 Eind venster](#).

2.3.2.2 Spelverloop moeilijkheidsgraad normaal & moeilijk

In deze graad worden weldegelijk alle functies gebruikt aangezien er nu wel een scorebord is waardoor de tijd moet worden bijgehouden van de gebruiker. De implementatie wordt hieronder gegeven via het UML diagram :

De code begint wederom bij de default constructor met het inladen van de geluidseffecten net zoals bij graad makkelijk. Met als verschil dat nu ook de SetTimer methode zal worden aangesproken. Waardoor .NET klasse DispatcherTimer wordt geïnitieerd dit zal zorgen voor de 5 seconden klok op het venster. Ook de .NET klasse Stopwatch wordt geïnitieerd, dit om de tijd bij te houden die de gebruiker nodig heeft om de 10 rekensommen op te lossen. Nu heeft de gebruiker wederom de keuze om uitleg te krijgen met de specifieke uitleg afhankelijk van de moeilijkheidsgraad. Of via de drukknop het spel te starten, waarbij de voortgangsbalk tevoorschijn komt en moeilijkere, grotere rekensommen gegeven worden. Maar nu worden ook de DispatcherTimer en Stopwatch gestart. De DispatcherTimer zal iedere seconde via een Tick event de methode TimerTick aanspreken. Dit zal er voor zorgen dat de 5 seconden klok zal aftellen. Wanneer de klok nog minder dan 3 seconden bedraagt zal het kleur van de klok veranderen in rood in plaats van zwart.

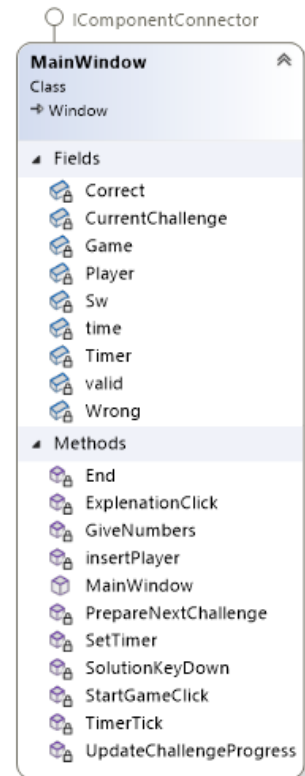
Indien de tijd verstreken is moet de volgende rekensom gegeven worden en krijgt de gebruiker geen punt. Dezelfde implementatie wordt voor de rest van het spel gebruikt zoals uitgelegd in de moeilijkheidsgraad makkelijk.

Totdat de gebruiker zijn laatste rekensom heeft opgelost. Dan zal eerst de Stopwatch stoppen met de tijd bij te houden. Enkel nu hebben we alle gegevens die we kunnen doorgeven aan de database voor het scorebord aan te vullen op het geschikte moeilijkheidsgraad. Dit wordt verwezenlijkt via de database DLL dat geschreven is in unmanaged C++ code te importeren in de managed C# code en de functie insertPlayer hieruit te halen. Onderstaande syntax wordt gebruikt:

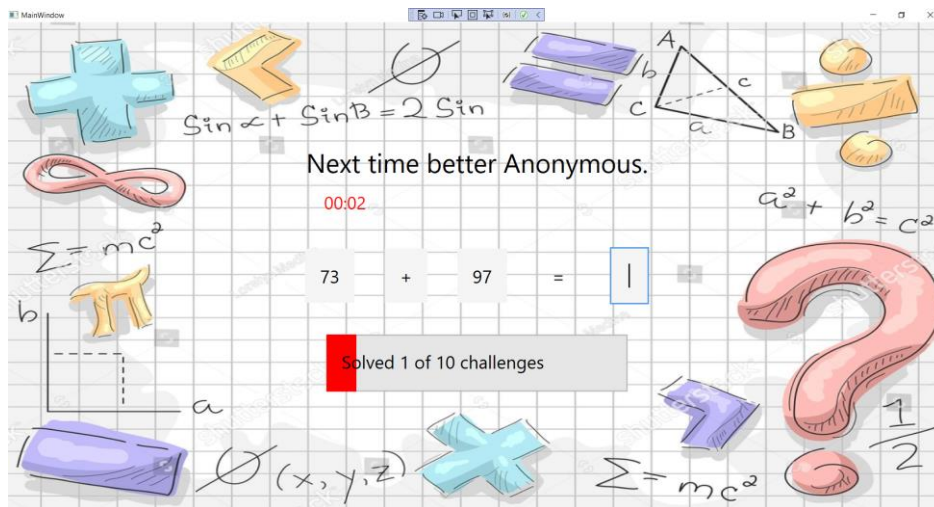
```
[DllImport(@"..\..\..\x64\Debug\DataLayer.dll", CallingConvention = CallingConvention.Cdecl)]  
2 references  
private static extern void insertPlayer(string table, string username, int score, double time);
```

In deze methode zal de naam, score, tijd en moeilijkheidsgraad waarin de gebruiker gespeeld heeft meegegeven worden als paramaters. Nadat deze gegevens zijn ingeladen in de database wordt het eindscherm aangesproken op dezelfde manier als besproken in [moeilijkheidsgraad makkelijk](#).

Op de volgende pagina wordt ook de lay-out meegegeven.



Hieronder wordt de lay-out weergegeven op moeilijkheidsgraad moeilijk, waarbij er duidelijk grotere rekensommen worden gegeven met de klok die aan het aftellen is:



2.3.3 Eind venster

De implementatie van het eind venster is afhankelijk van welke moeilijkheidsgraad de gebruiker heeft gekozen om het spel te spelen. Dit omdat er bij moeilijkheidsgraad normaal en moeilijk een database aanwezig is, die moet worden weergegeven en bij graad gemakkelijk niet. Hierdoor zijn er 2 soorten eindschermen die hieronder verder zullen besproken worden.

2.3.3.1 Speleinde moeilijkheidsgraad makkelijk

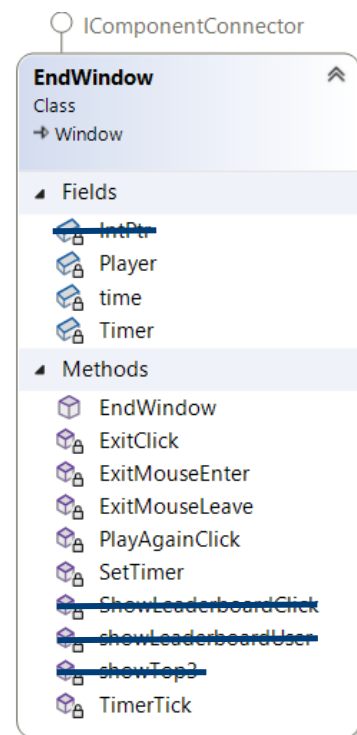
In deze graad wordt een overzicht gehouden van zowel de gevraagde rekensommen als de bijhorende oplossingen maar ook de totale score van de gebruiker wordt weergegeven.

Daarnaast krijgt de gebruiker de keuze om opnieuw een nieuw spel te spelen of om het spel te beëindigen. Hieronder wordt de gebruikte implementatie besproken.

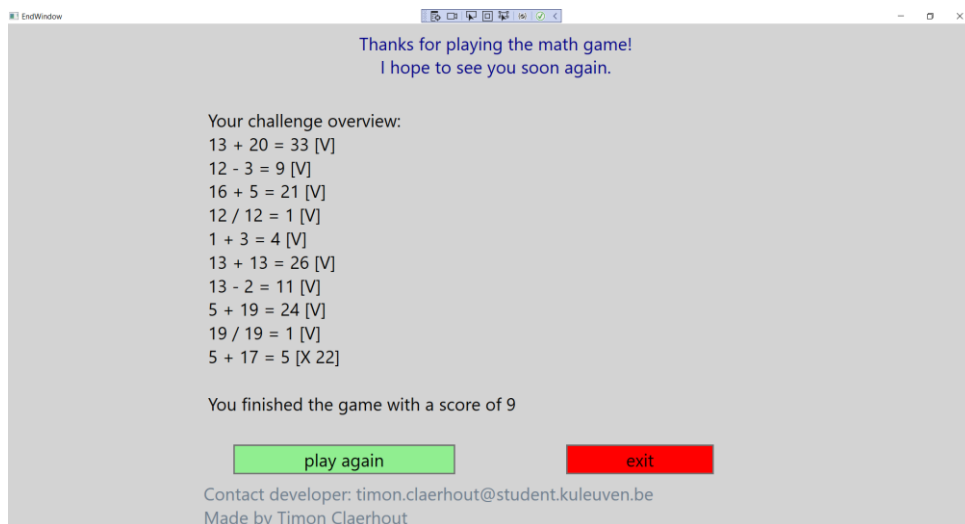
Zoals eerder vermeld wordt de tijd en database niet gebruikt, hierdoor zijn enkele functies geschrapt in onderstaand UML diagram.

De code begint uiteraard bij de default constructor EndWindow, hierbij zal het overzicht en score weergegeven worden zoals eerder besproken. Deze waarden krijgt hij via het hoofd venster mee als parameter. Er is ook een Exit knop voorzien om het spel te verlaten, indien de gebruiker met zijn muis over de knop gaat zal de methode ExitMouseEnter aangesproken worden. Deze methode zorgt er voor dat de tekst "Exit" verandert naar "Are you sure?". Wanneer de muis terug uit de knop gesleept wordt zal de ExitMouseLeave methode aangesproken worden. Deze zal de "Are you sure?" tekst terug aanpassen naar de originele "Exit" tekst. Wanneer de gebruiker op de exit knop klikt zal de ExitClick aangesproken worden waardoor de Math Game volledig zal stoppen. Er is ook een Play Again knop voorzien om een nieuw spel te starten. Om de gebruiker attent te maken om deze knop te gebruiken laten we deze knop knippen van kleur. Dit wordt verwezenlijkt door de TimerTick methode die iedere seconde aangesproken wordt, net zoals in het [spelverloop](#). Wanneer de speler hierop klikt wordt het login venster opnieuw gebruikt. Maar met als verschil dat de non-default constructor deze keer wordt gebruikt omdat de gebruikersnaam als parameter wordt meegegeven. Hierdoor moet de gebruiker niet opnieuw zijn/haar gebruikersnaam typen, maar blijft deze bewaard.

De lay-out van deze implementatie zal getoond worden op de volgende pagina.



Hieronder is de lay-out weergegeven die gebruikt wordt bij moeilijkheidsgraad makkelijk:

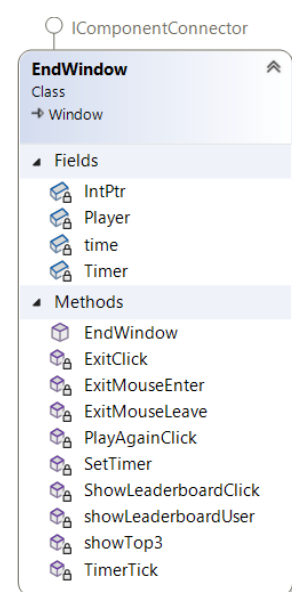


2.3.3.2 Speleinde moeilijkheidsgraad normaal & moeilijk

In deze graad wordt een overzicht gehouden van zowel de gevraagde rekensommen als de bijhorende oplossingen maar ook de totale score en de tijd van de gebruiker wordt weergegeven met de corresponderende plaats op het scorebord. Daaronder worden de top 3 spelers getoond van de corresponderende moeilijkheidsgraad.

Daarnaast krijgt de gebruiker de keuze om opnieuw een nieuw spel te spelen, het gehele scorebord te bekijken of om het spel te beëindigen. Hieronder wordt de gebruikte implementatie besproken.

Dezelfde werkwijze wordt gehanteerd zoals besproken in de vorige moeilijkheidsgraad. Maar met als verschil wordt er een scorebord getoond. De tijd van de speler wordt meteen doorgegeven vanuit het hoofd venster, dus zijn er geen extra methodes vereist hiervoor. Voor de gegevens uit de databank te halen zijn er wel extra methodes voorzien. Dit door de DLL functies te importeren, maar dit gaat niet zomaar. Doordat deze functies een string moeten teruggeven van het scorebord via de unmanaged C++ code (dat een `char*` gebruikt) naar de managed C# code (dat een string gebruikt). Doordat de managed code geen idee heeft hoe het geheugen werd toegewezen in de unmanaged code, weet hij niet hoe hij deze terug moet vrij maken wat zal resulteren in een foutcode. Om dit probleem op te lossen wordt .NET IntPtr klasse gebruikt.



Dit zal de unmanaged char* automatisch omzetten naar een managed Unicode String-object. Onderstaande syntax wordt gebruikt om de showTop3 en showLeaderBoardUser methode te importeren en de string uit te lezen:

```
private IntPtr IntPtr;  
[DllImport(@"..\..\..\x64\Debug\DataLayer.dll", CallingConvention = CallingConvention.Cdecl)]  
2 references  
private static extern IntPtr showTop3(string table);  
  
IntPtr = showLeaderboardUser("Normal_Leaderboard", Player.Username, Player.Time);  
LeaderboardUser.Text = $"{Marshal.PtrToStringAnsi(IntPtr)}";
```

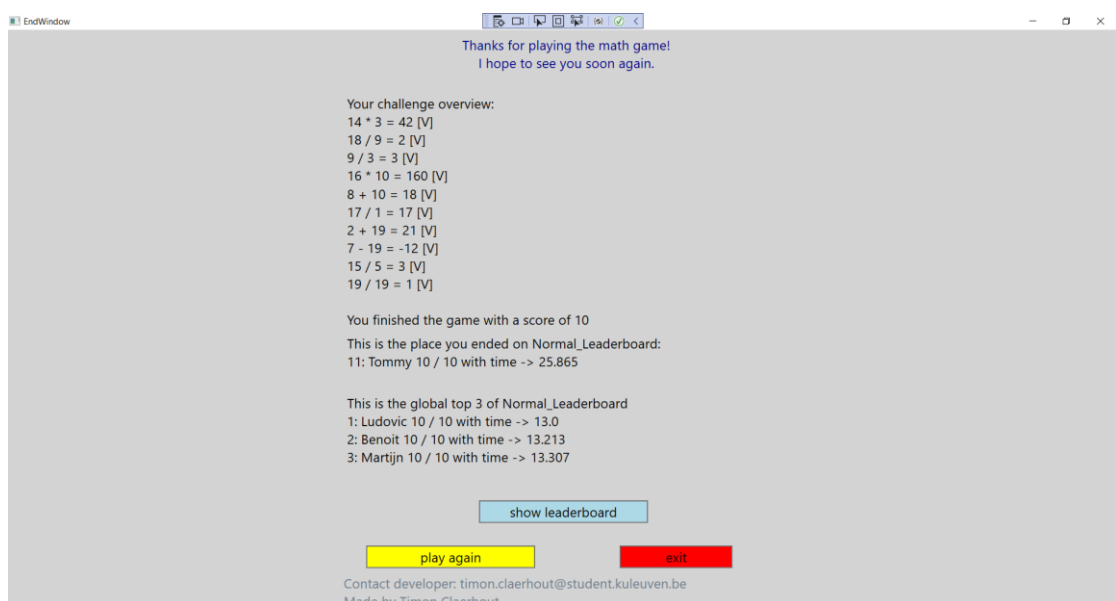
Op deze manier wordt na het weergeven van het speloverzicht van de rekensommen eronder de methode showLeaderBoardUser gebruikt. Dit zal de gepresteerde prestaties van de gebruiker weergeven met de corresponderende plaats. Deze plaats wordt automatisch uit de database gehaald door te filteren op zowel de desbetreffende gebruikersnaam als de tijd. Aangezien de tijd op 0.001 seconden nauwkeurig wordt bijgehouden, is dit een geschikte methode om te filteren in de gehele database.

Daaronder zal showTop3 via dezelfde syntax gebruikt worden om de top 3 spelers weer te geven van de specifieke moeilijkheidsgraad.

Daar nog eens onder is er een knop Leaderboard voorzien zodat het gehele scorebord kan bekeken worden van de corresponderende moeilijkheidsgraad. Dit gebeurt op het [scorebord venster](#) dat op de volgende pagina verder zal besproken worden.

Ten slotte gaan de knoppen Exit en Play Again zal op dezelfde manier geïmplementeerd worden zoals bij [graad makkelijk](#) vermeld.

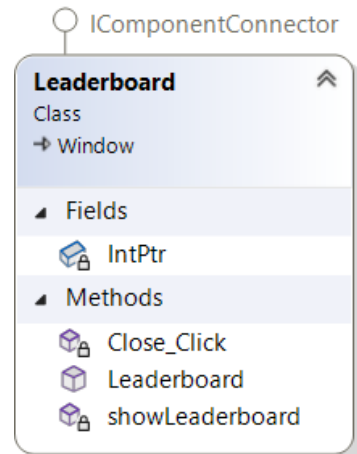
Hieronder is de lay-out weergegeven van de besproken implementatie:



2.3.4 Scorebord venster

Wanneer de gebruiker op moeilijkheidsgraad normaal of moeilijk speelt, krijgt de gebruiker de keuze om het gehele scorebord te bekijken via het eind venster wanneer er op de Leaderboard knop geklikt wordt. Hierdoor zal het scorebord venster aangesproken worden wat hieronder verder zal toegelicht worden via het UML diagram.

De code start uiteraard bij de Leaderboard constructor, deze zal de showLeaderboard functie oproepen om het gehele scorebord weer te geven op specifieke moeilijkheidsgraad via een “TextBox” met scroll bar. Deze functie is wederom geïmporteerd via de database DLL met behulp van de IntPtr klasse, dit op dezelfde manier zoals besproken op de vorige pagina. Onder het scorebord is een knop voorzien om het venster te sluiten, wanneer er geklikt wordt op de knop zal de functie Close_Click aangesproken worden. Deze zal het venster sluiten waardoor de gebruiker terug op het eind venster terecht komt.



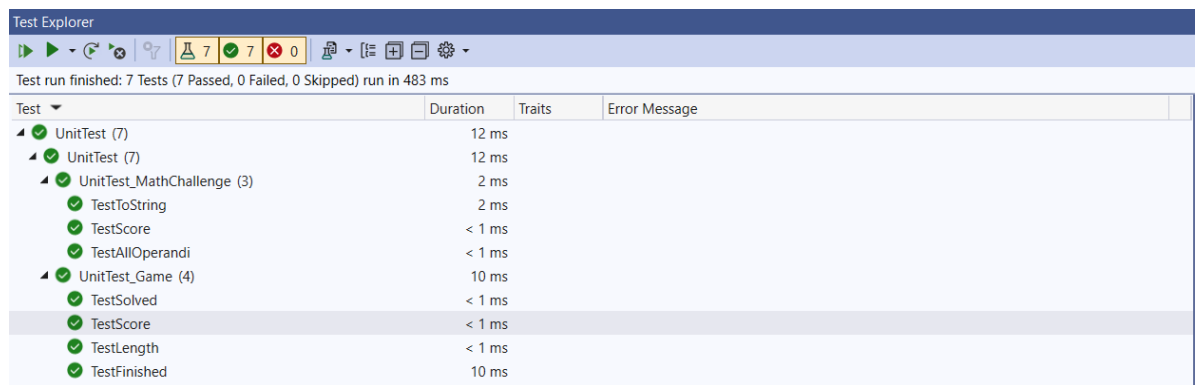
Hieronder is de lay-out gegeven van de besproken implementatie:



3 Unit test

Het testen van code is zeer belangrijk en zorgt ervoor dat eventuele problemen gemakkelijker opgespoord raken. Daarom hebben ik enkele testen gemaakt die code uittest. Specifiek worden de functies uit de klassen MathChallenge en Game uitgetest. De Player klasse wordt overgelaten aangezien deze enkel statistieken bijhoudt van de gebruiker wat amper testimplementatie nodig heeft. De testen vullen enkele waarden in de functies en controleren dan als deze overeenkomen met zelf berekende waarden. Wanneer ik al deze testen laat uitvoeren zijn alle testen correct uitgevoerd. Wat er toe leidt dat de code doet wat er verwacht wordt, waardoor de correcte werking wordt bewezen.

Dit kan je zien op onderstaande afbeelding:



Test	Duration	Traits	Error Message
✔ UnitTest (7)	12 ms		
✔ UnitTest (7)	12 ms		
✔ UnitTest_MathChallenge (3)	2 ms		
✔ TestToString	2 ms		
✔ TestScore	< 1 ms		
✔ TestAllOperandi	< 1 ms		
✔ UnitTest_Game (4)	10 ms		
✔ TestSolved	< 1 ms		
✔ TestScore	< 1 ms		
✔ TestLength	< 1 ms		
✔ TestFinished	10 ms		

4 Bijlagen & bronnen

- Hoe implementeer je een countdown:

https://www.youtube.com/watch?v=o_F_v_ISeDk

- Hoe implementeer je een timer:

<https://www.codeproject.com/Articles/98346/Microsecond-and-Millisecond-NET-Timer>

- Basis van WPF styling:

<https://www.youtube.com/watch?v=Vjldip84CXQ&t=2044s>

- Hoe geluidseffecten afspelen:

<https://www.youtube.com/watch?v=ddmUo3YMfzg>

- Gebruikte .WAV geluidseffecten:

<https://pixabay.com/sound-effects/>

- String versturen van unmanaged C++ DLL naar managed C# en terug:

<https://www.codeproject.com/Articles/1189085/Passing-Strings-Between-Managed-and-Unmanaged-Code>

- Converteren van string naar char*:

<https://www.techiedelight.com/convert-std-string-char-cpp/>

- SQLite3 source code (gebruikte bestanden sqlite3.h, shell.c en sqlite3.c):

<https://sqlite.org/download.html>

AFDELING
Straat nr bus 0000
3000 LEUVEN, België
tel. + 32 16 00 00 00
fax + 32 16 00 00 00
@kuleuven.be
www.kuleuven.be

