

RRIQ Design document

Gemaakt door:

Renas Khalil

Versie: 1.5

Datum: 03-04-2024

Inhoudsopgave

- Inhoudsopgave (2)
- Belangrijke rollen (3)
- Plan van Eisen (4)
- Huisstijl (5)
- Wireframe (6,7,8)
- Mock up (9,10,11,12,13)
- Definition of Done (14)
- User stories (15,16)
- UML (17)
- ERD (Entity Relationship Data model) (18,19)
- UML diagrammen (20,21,22,23,24)
- Solid (25,26)
- Desing Smells (27)
- Desing Patterns (28,29,30,31,32)
- Sources (33)
- Nawoord (34)

Belangrijke rollen

The Product Owner

Jan Zuur

Tom Sivers

Ron van Zuilichem

SCRUM Master

Renas Khalil

DEV Team

Renas Khalil

Quinten Verf

Radjen Ellis

Plan van Eisen

Belangrijkste Kenmerken

De applicatie zal gebruikmaken van object georiënteerd programmeren in C# om de code te structureren.

De belangrijkste functionaliteiten van de applicatie zelf zijn: Game opstarten, pong spelen en de highscores zien

Game opstarten

Gebruikers moeten de console app kunne opstarten en een sessie pong tegen een computer kunnen spelen en afmaken.

game spelen

Gebruikers moeten hun pallet kunnen bewegen om de pong bal terug te kaatsen naar de computer en proberen te scoaren voor punten.

Gebruikers moeten hun goal kunnen verdedigen door het bewegen van de pallet

Highscores

Gebruikers hun gegevens moeten na een complete sessie van pong opgeslagen worden in een database en weergaven worden in de highscores pagina.

De highscores moet gesoteerd worden van hoog naar laag om te laten zien we de beste scores heeft behaald tegen de computer

Algemene doelen

Het bouwen van een gebruiksvriendelijke console applicatie die het spel pong kan spelen

Het ontwikkelen van een database die data opslaat van de gebruiker.

Het testen en valideren van de applicatie om ervoor te zorgen dat deze voldoet aan de gestelde eisen en normen.



Huisstijl

Color pallet / Font

De kleuren pallet voor de Pong Applicatie zal voor het algemeen donkere tinten zijn en we zullen gebruik maken van Gebruikte Kleuren: Pink RGB: 252 / 236 / 236 en Navy Blue RGB: 51 / 61 / 121 en Black: 0 / 0 / 0

Main color:  Navy Blue RGB: 51 / 61 / 121

Buttons + scoreboard  Pink RGB: 252 / 236 / 236

Text:  Black RGB: 0 / 0 / 0 +  Pink RGB: 252 / 236 / 236

Font: Default font gebruikt.

We zien hier een lichtroze kleur met een tint marineblauw. Het contrast is heel opvallend, maar de twee kleuren vullen elkaar prachtig aan. Deze combinatie zorgt voor een rustig en netten/strakke omgeving voor de applicatie.

Sources:

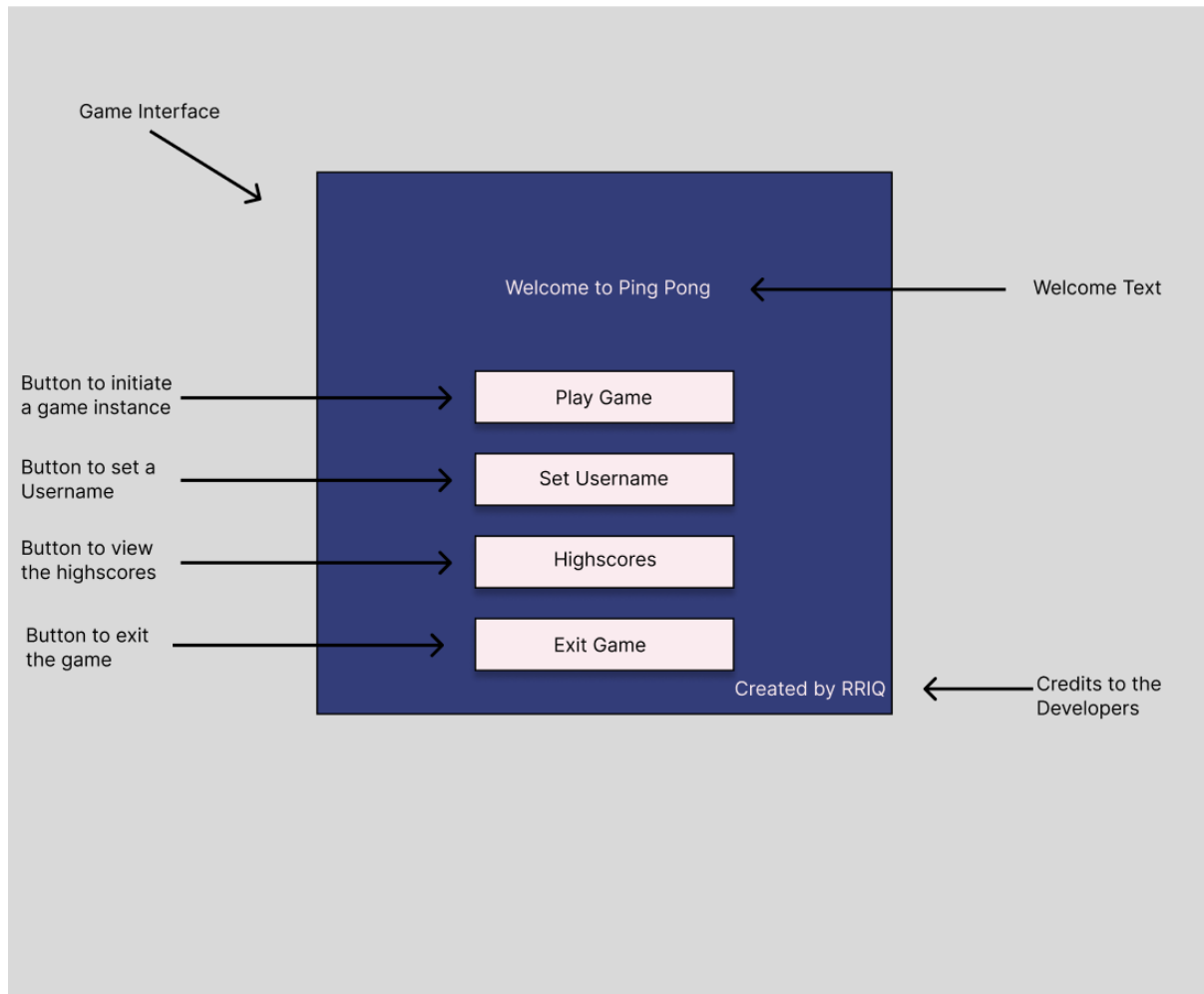
<https://designwizard.com/blog/colour-combination/> Voor de kleur combinatie

<https://imagecolorpicker.com/> Hex + RGB Picker

Wireframe

De wireframe bestaat uit de volgende pagina's:

Home Menu, Laat de home pagina van de pong game zien.



Welcome text: Een stuk text dat de gebruiker verwelkomt tot de game

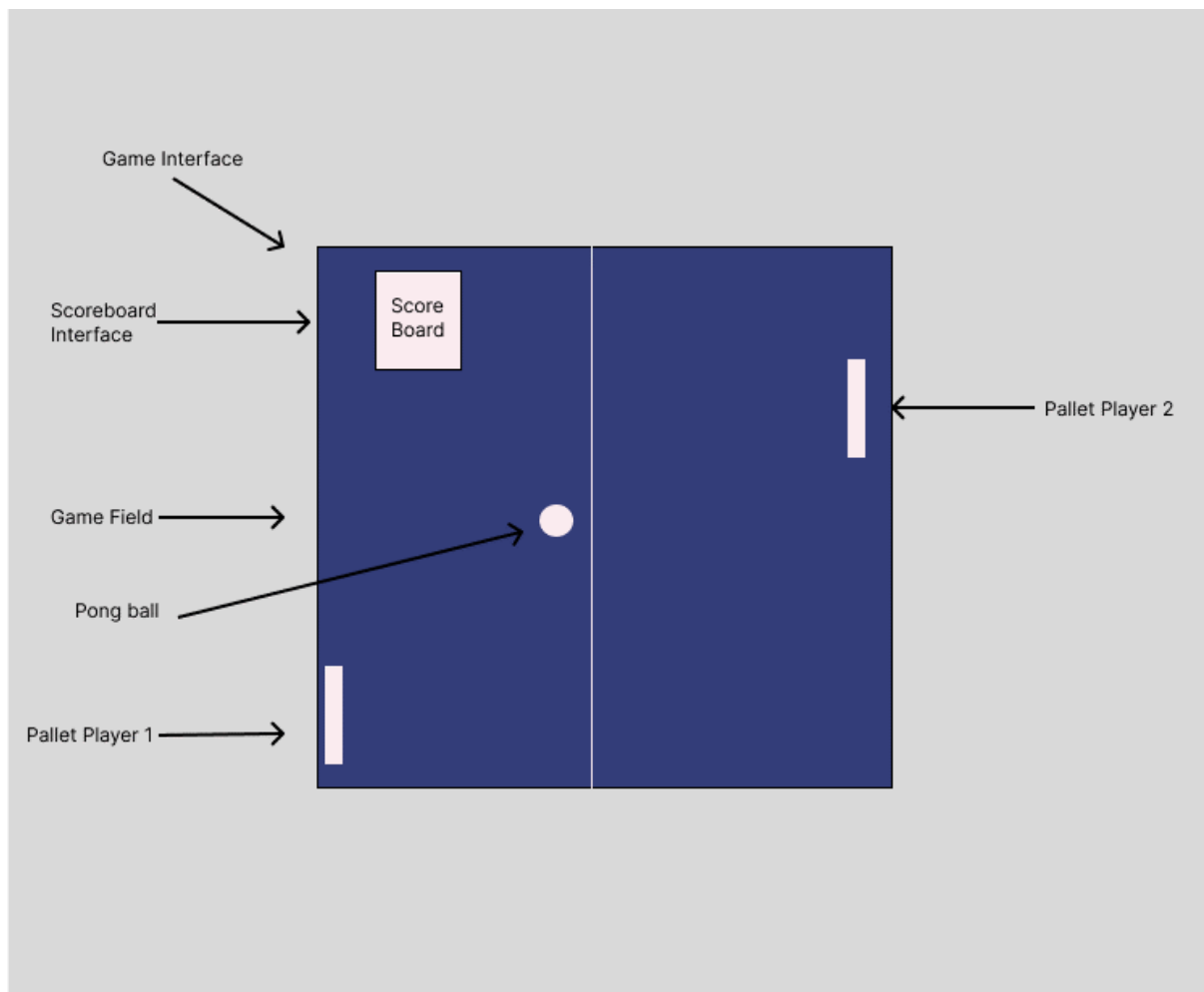
Game interface: De spelinterface waar de inhoud van de spel bevint

Buttons / Knoppen:

- Play game: Start een sessie pong op
- Set Username: De gebruiker kan een naam zetten
- Highscores: Gaat naar de topscores van de game
- Exit Game: Verlaat de spel

Developer credits: Laat zien welke team het spel heeft ontwikkeld

Game interface, via de “play game” knop navigeer je naar de spel interface.



Game interface: Het spel interface waar de pong game in gespeeld wordt

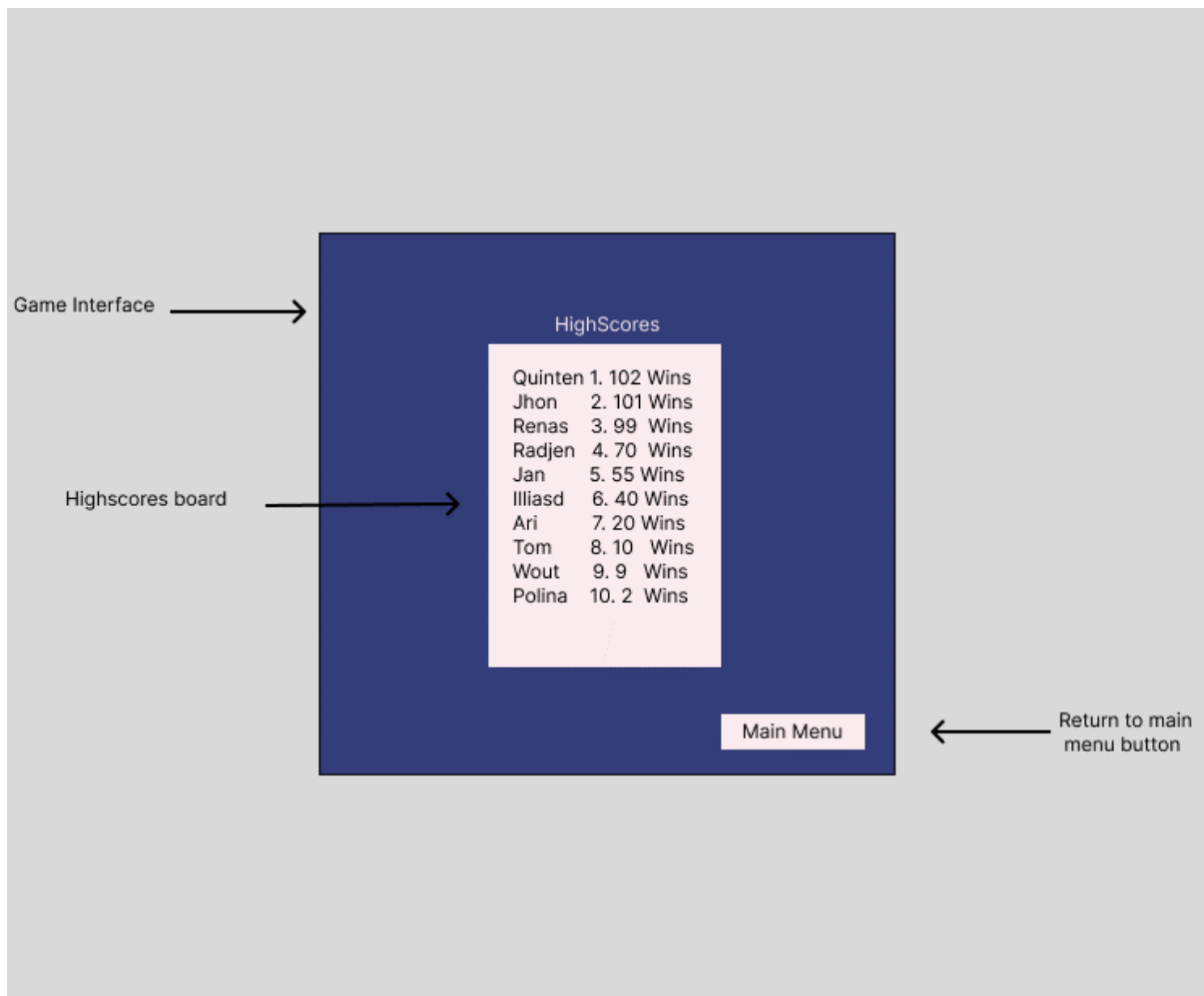
Scoreboard interface: Laat de huidige score zien van de speler

Game field: Het veld waar de objecten van het spel in zitten

Pong ball: De pong ball die heen en weer wordt gekaats tot een van de players scoort

Pallet player 1 / 2: De pallet die de speler beweegt om bal te kaatsen naar de tegenstander.

Leaderboard, Hier wordt de scores van de spelers opgeslagen en weergaven.



Game interface: De interface waar de leaderboards in word getoond

Highscores board: De scoredboard waar alle scores worden bijgehouden en genummerd + gestoreerd wordt

Main menu knop: De knop die je terug breng naar de Home menu.

Mock up

Mock up pingpong

TEAM Rriq

Legenda

0 welkom tekst

1 start game

2 kies username

3 highscore

4 verlaat het spel

5 paddle player 1

6 paddle player 2

7 pong

8 veld

9 scoreboard

10 achtergrond

11 blauwe achtergrond bij Highscores

12 de highscore lijst

13 terug naar Menu t de leaderboard

14 indicator waar je bent en waar je bent

15 waar je de username moet zetten

16 speel het spel

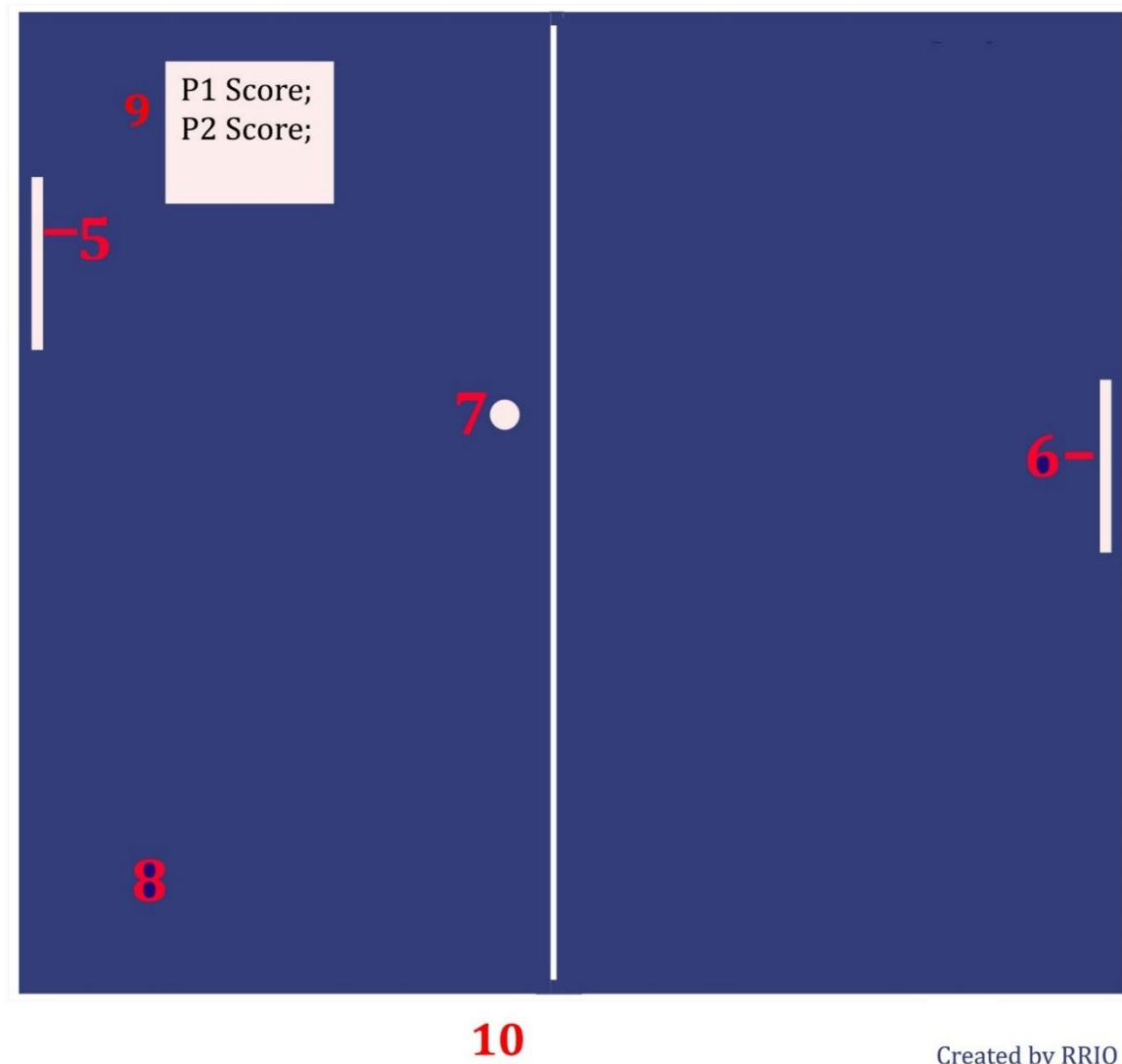
Main Menu Mock-up

1. Verwelkomt nieuwe of terugkerende spelers voor hun nieuwe avontuur
2. Als je op deze knop drukt start het spel
3. Als je op deze knop drukt dan ga je naar het scherm waar de speler een username kan aanmaken
4. Hier kan de speler de highscores zien
5. om de game te verlaten



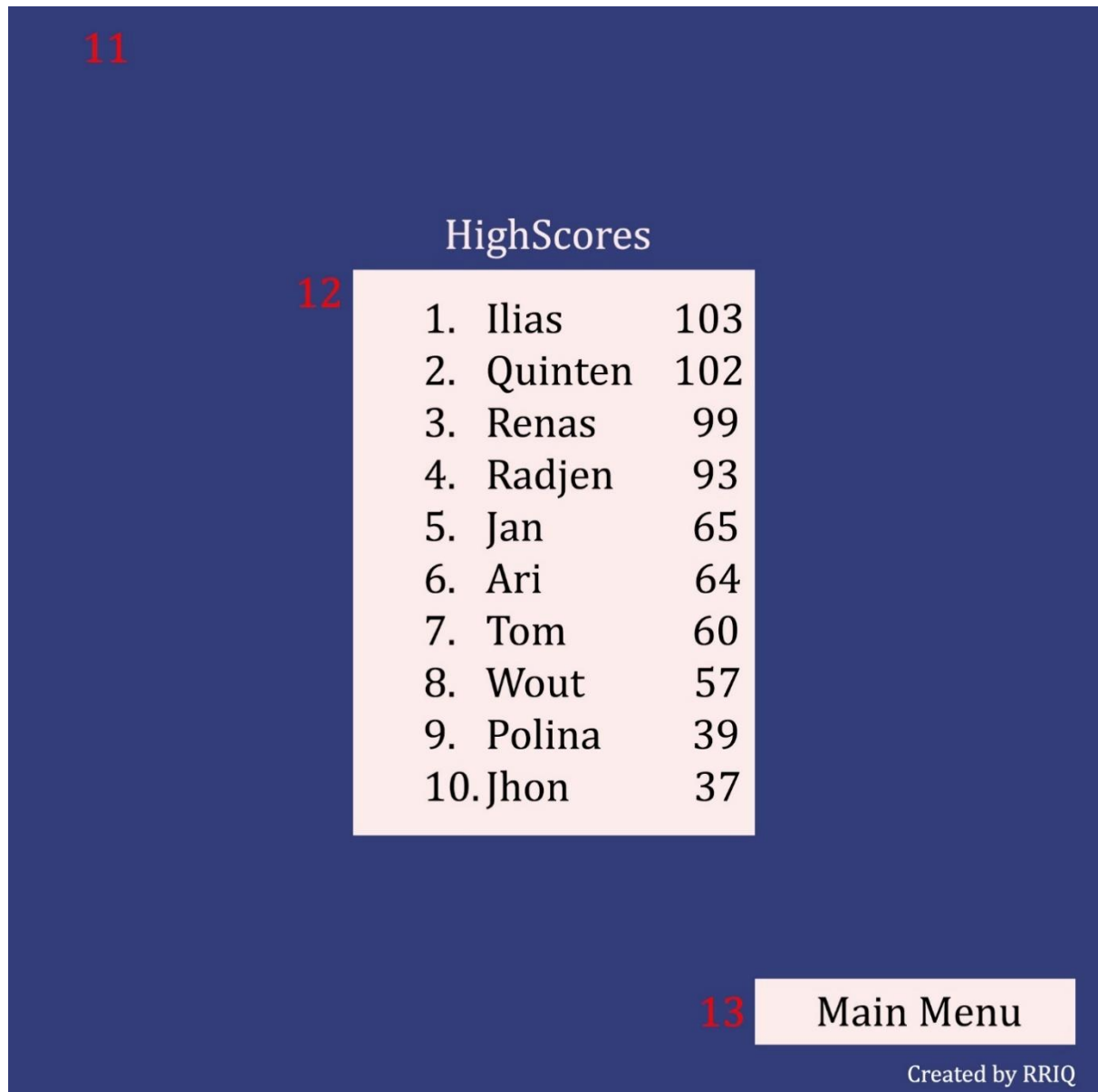
Game interface Mock-up

6. Paddle van player 1 (linkerkant van het veld)
7. De paddle van player 2 (rechterkant)
8. De pong waarmee gespeeld word
9. Het speelveld hier wordt bijgehouden wie voor staat en met hoeveel
10. Buiten het veld staat een wit achtergrond



Highscores Mock-up

11. Blauwe achtergrond
12. Scoren bord hier kan je zien wie de highscore heeft
13. Terug gaan naar de Main menu



Set username Mock-Up

14. Hier kan je zien dat je in de set username pagina bent
15. Hier kan je de username type dan word je score bijgehouden
16. Begin het spel



Definiton of Done

Stap 1: Code voldoet aan de eisen van de userstorie.

Stap 2: Code is duidelijk gecoment.

Stap 3: Code is getest en heeft geen bugs in de branch.

Stap 4: Code review gehouden in de pull request.

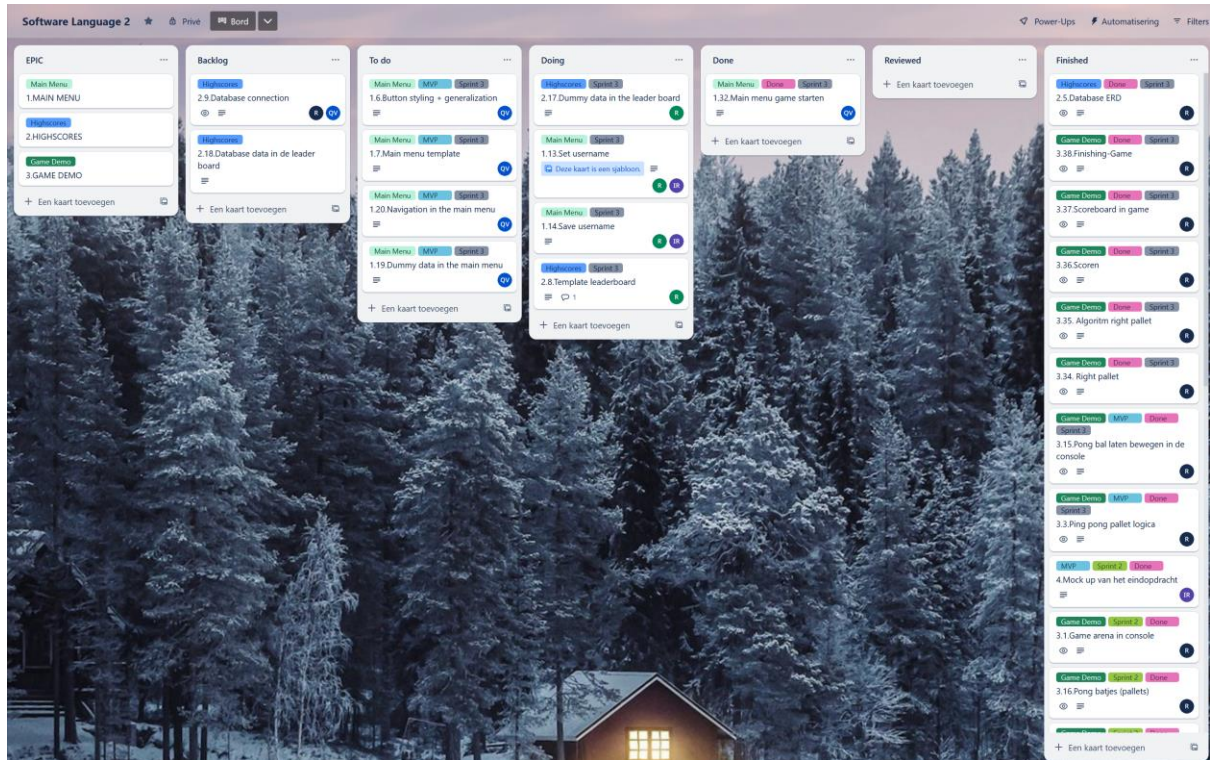
Stap 5: Code mag gepushed worden naar de main branch

Stap 6: Ticket word verplaats naar Done op de trello bord

Stap 7: Vertel de team dat de ticket klaar is en ga over de veranderingen met de team zodat iedereen up to date is.

Userteries

Voor de usertories hebben wij gekozen om Trello te gebruiken als onze scrumboard. Dit is een voorbeeld van de scrumboard van 03-04-2024.



In de scrumboard zullen de userstories er zo uitzien:

4.Mock up van het eindopdracht

in lijst Doing

Leden

R

+

Labels

MVP

Sprint 2

+

Meldingen

Volgen

✓

Omschrijving

Bewerken

Als ontwikkelaar wil ik de mock up hebben, zodat we weten hoe de pong spel eruit zal zien

Activiteit

Verberg details

R

Maak een opmerking...

13.Set username

in lijst Backlog

Meldingen

Volgen

Omschrijving

Bewerken

Als gebruiker wil ik mijn naam kunnen zetten in de game, zodat ik weet welke score bij mij hoort

Activiteit

Verberg details

R

Maak een opmerking...

20.Navigation in the main menu

in lijst Backlog

Meldingen

Volgen

Omschrijving

Bewerken

Als gebruiker wil ik in de main menu kunnen navigeren, zodat ik een game pong kan starten of de highscores kan zien

1.Game arena in console

in lijst Doing

Leden

R

+

Labels

Sprint 2

+

Meldingen

Volgen

✓

Omschrijving

Bewerken

Als ontwikkelaar wil ik een vierkante arena hebben, zodat ik dit als speelveld kan gebruiken.

Activiteit

Verberg details

R

Maak een opmerking...

5.Database ERD

in lijst To do

Leden

R

+

Labels

Sprint 2

+

Meldingen

Volgen

✓

Omschrijving

Bewerken

Als ontwikkelaar wil ik een ERD hebben, zodat ik de datastructuur kan inzien van de database

Activiteit

Verberg details

R

Maak een opmerking...

14.Save username

in lijst Backlog

Meldingen

Volgen

Omschrijving

Bewerken

Als gebruiker wil ik mijn naam kunnen opslaan, zodat ik later terug kan kijken wat mijn scores was in de highscores

De EPICS van de pong applicatie zijn als volgende:

EPIC: 1.Main menu en de bijhordende tickets:

1. 1.6. Button styling + generalization
2. 1.7. Main menu template
3. 1.13. Set username
4. 1.14. Save username
5. 1.19. Dummy data in the main menu
6. 1.20. Navigation in the main menu
7. 1.32.Main menu game starten

EPIC: 2.High scores en de bijhordende tickets:

1. 2.5. Database ERD
2. 2.8. Template Leaderboard
3. 2.9. Database Connection
4. 2.17. Dummy data in the leader board
5. 2.18. Database data in de leader board

EPIC: 3.Game demo en de bijhordende tickets:

1. 3.1. Game arena in console
2. 3.2. Pong bal in console
3. 3.3. Ping pong pallet logica
4. 3.15. Pong bal laten bewegen in de console
5. 3.16 Pong batjes (pallets)
6. 3.34. Right pallet
7. 3.35. Algoritm right pallet
8. 3.36. Scoren
9. 3.37. Scoreboard in game
10. 3.38. Finishing-Game

De userstories zijn met de hulp van https://www.wikiwand.com/en/User_story#Usage gemaakt, hiervoor heb de template **Connextra template** gekozen.

UML

UML staat voor Unified Modeling Language, het is een gestandaardiseerde modelleertaal die wordt gebruikt om de gedrag en architectuur van een softwaarsysteem te visualiseren, specificeren ontwerpen en documenteren. UML maakt gebruik van verschillende soorten diagrammen om dit doel te bereiken.

Source: <https://www.lucidchart.com/pages/nl/wat-is-unified-modeling-language>

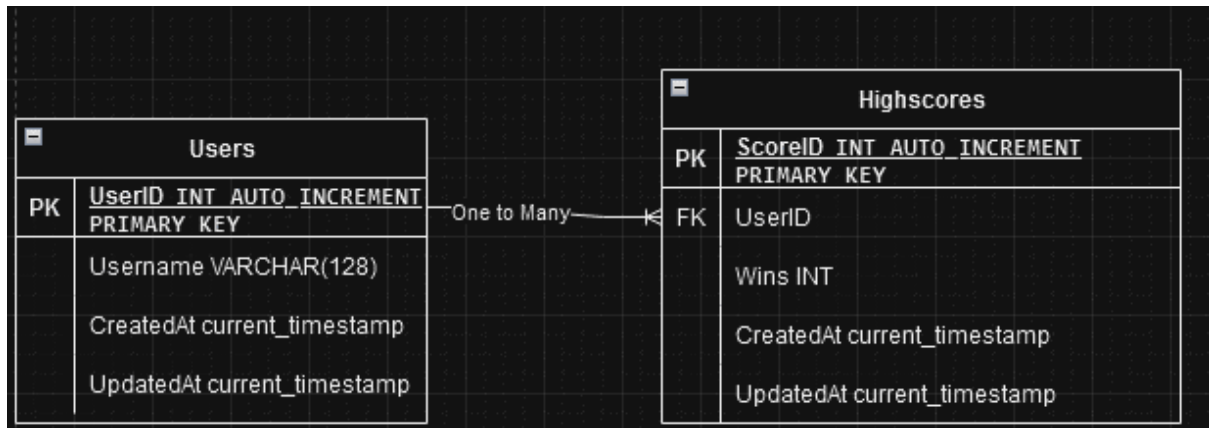
Wij maken gebruik van **UML Diagrammen** voor het vastleggen van de structuur en het gedrag van het systeem.

Voor de pong app zelf maken wij gebruik van de volgende Diagrammen:

1. **Entity Relationship Datamodel (ERD)**
2. **Activity diagram**
3. **Use case diagram**
4. **Sequence diagram**
5. **Klassen diagram**
6. **Component diagram**

ERD

De Entity relationship data model van de console pong app willen we dat de gebruiker zijn username kan opslaan en zijn behaalde scores in de database worden opgeslagen. Aan de hand van Mysql queries naar de highscores toe kunnen we sorteren op de meeste wins en laten zien welke gebruikers deze heeft.



We hebben 2 tables, een user table en een highscores table.

De user table bestaat uit:

1. UserID AUTO_INCREMENT PRIMARY KEY
2. Username VARCHAR (128)
3. CreatedAt Current_timestamp
4. UpdatedAt Current_timestamp

Uitleg van de velden:

UserID: Voor elke gebruiker slaan we een unique userID, dit doen we zodat we elk gebruiker kunnen identificeren aan hun ID.

Username: We slaan de naam op van de user in de tabel

CreatedAt: Slaat de tijd op wanneer een van de velden worden geupdate met de volgende formaat: 2024-04-1 22:49:44

UpdatedAt: Slaat op Wanneer de user word geupdate bijvoorbeeld wanneer de naam word veranderd met de volgende formaat: 2024-04-1 22:49:44

De HighScores table bestaat uit:

1. ScoreID AUTO_INCREMENT PRIMARY KEY
2. UserID Foreign Key(FK)
3. Score INT NOT NULL
4. CreatedAt Current_timestamp
5. UpdatedAt Current_timestamp

Uitleg van de velden:

ScoreID: Voor elke score die wordt behaald slaan we een unieke scoreID op, dit doen we zodat we elk behaald score kunnen identificeren aan hun ID.

UserID: Wordt als Foreign Key opgeslagen met de behaalde score, op deze manier slaan we de score op en linken we deze aan de user die dit heeft behaald.

Wins: Slaat op hoe vaak de gebruiker heeft gewonnen, we sorteren ook op deze veld om te laten zien wie nummer 1 is met de meeste wins

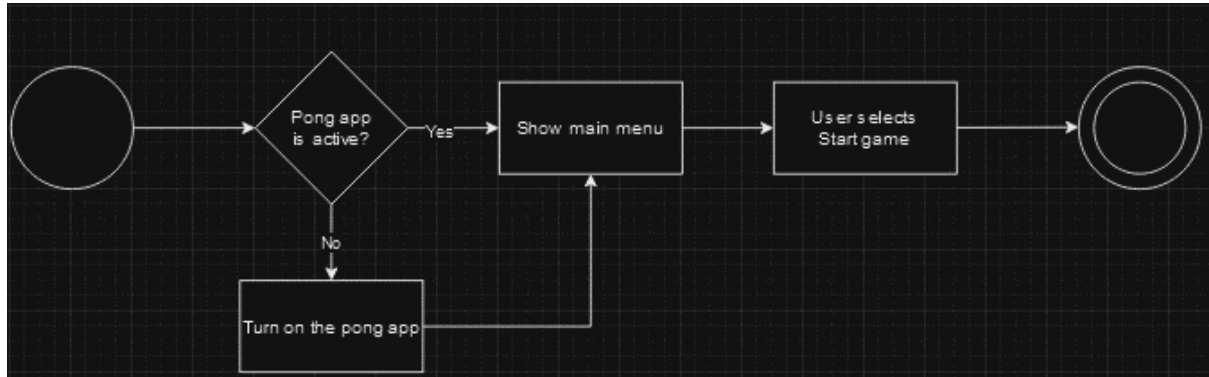
CreatedAt: Slaat op wanneer de score is gemaakt met de volgende formaat: 2024-04-1 22:49:44

UpdatedAt: Slaat de tijd op wanneer een van de velden worden geupdate met de volgende formaat: 2024-04-1 22:49:44

UML Activity Diagram

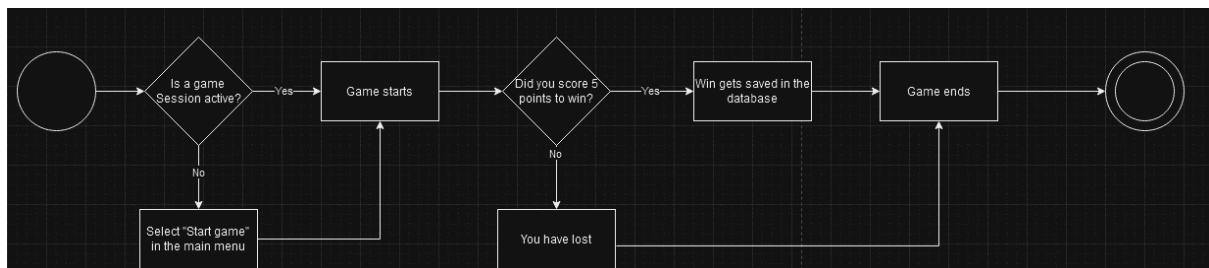
Activity diagram: activiteitschema is een diagram dat het verband weergeeft tussen verschillende activiteiten of bewerkingen meestal van een proces binnen een bedrijf.

Start a game diagram: Laat zien hoe een game word gestart



We beginnen met een beslis moment waar we controleren of de gebruiker de pong applicatie actief heeft. Als het actief is laten we de main menu van de pong applicatie zien. Zoniet dan starten we deze eerst op voor dat de gebruiker de main menu kan zien. In de main menu selecteerd de user "Startgame" om een game te starten, hier eindigt het proces ook.

Game session diagram: Laat zien hoe een sessie pong eruit ziet

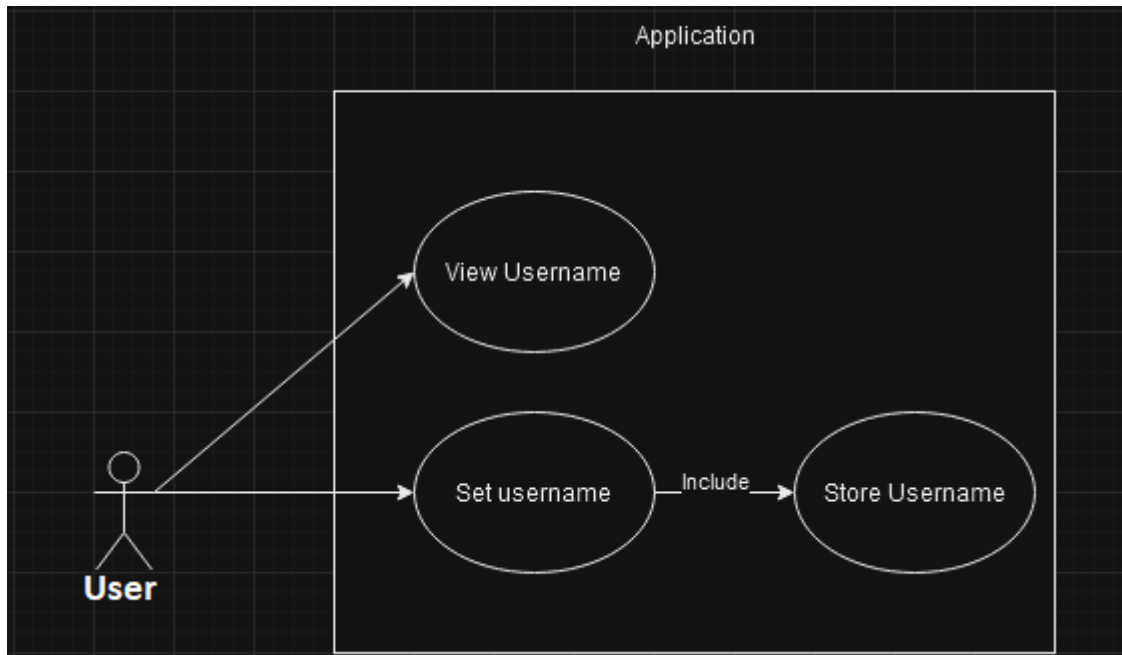


We beginnen het proces met het checken of de gebruiker al een game heeft gestart, zo niet dan laten we gebruiker in the main menu de "start game" knop drukken op een game te starten. Dit begint een spell vervolgens kijken we of de gebruiker 5x heeft gescoord. Zoja dan heeft de gebruiker gewonnen en word zijn win in de database opgeslagen en word de game beindigt.. Zoniet dan heeft de gebruiker verloren en eindigt de game.

UML Use Case Diagram

Use Case Diagram: Laat zien hoe verschillende gebruikers (actoren) interactie hebben met het systeem. Bijvoorbeeld: gebruiker ziet username, gebruiker voegt username toe, etc.

Set username diagram: Laat zien hoe een user zijn naam zet of ziet en hoe de systeem hiermee omgaat.



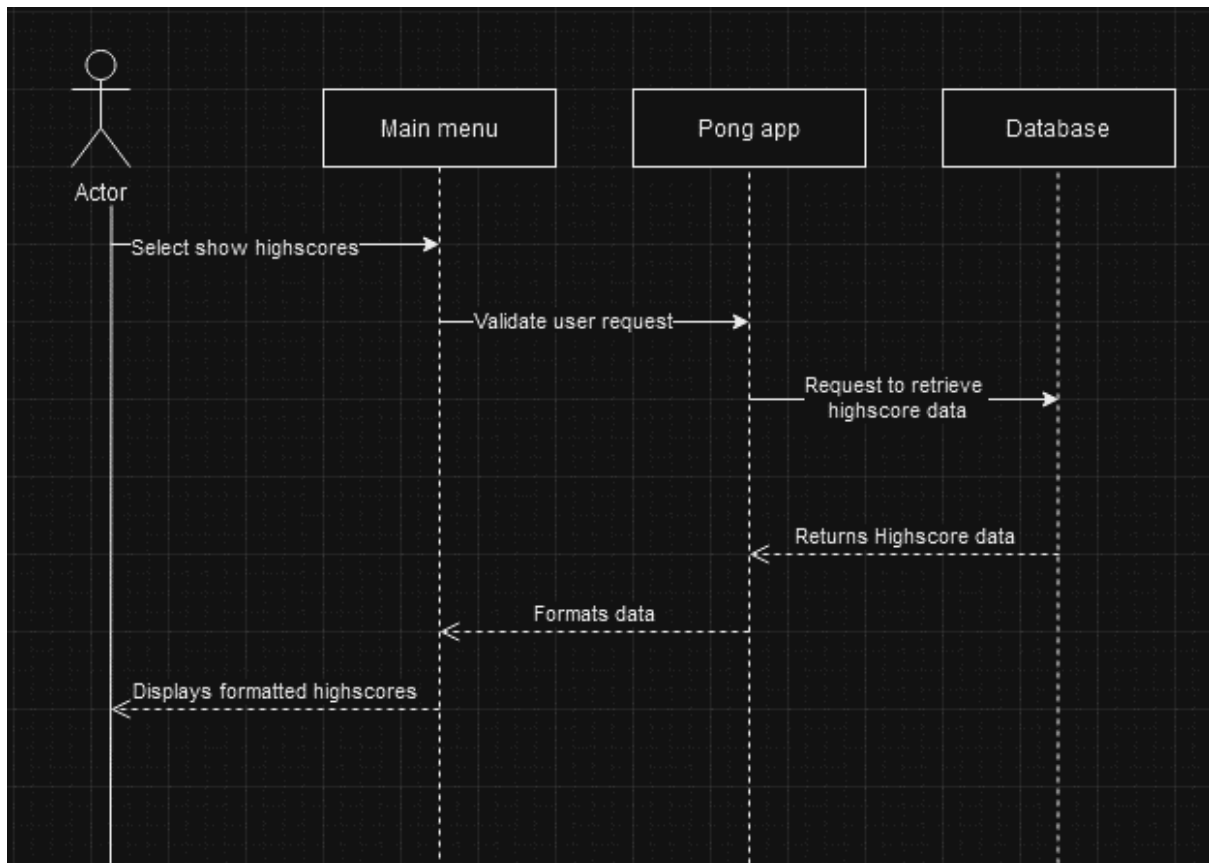
De gebruiker(actor) bevindt zich in de applicatie. Hierin kan de gebruiker vervolgens 2 hoofdacties uitvoeren. Set username, en view username. Set username laat de actor een username typen die vervolgens word opgeslagen in de database. En view laat de actor zijn huidige username zien.

UML Sequence Diagram

Sequence Diagram: Beschrijft de interactie tussen objecten in een sequentiële volgorde.
Bijvoorbeeld: de stappen die plaatsvinden wanneer een gebruiker een highscore wilt zien.

Sequence diagram Highscores: Laat zien hoe een gebruiker de highscores krijgt te zien.

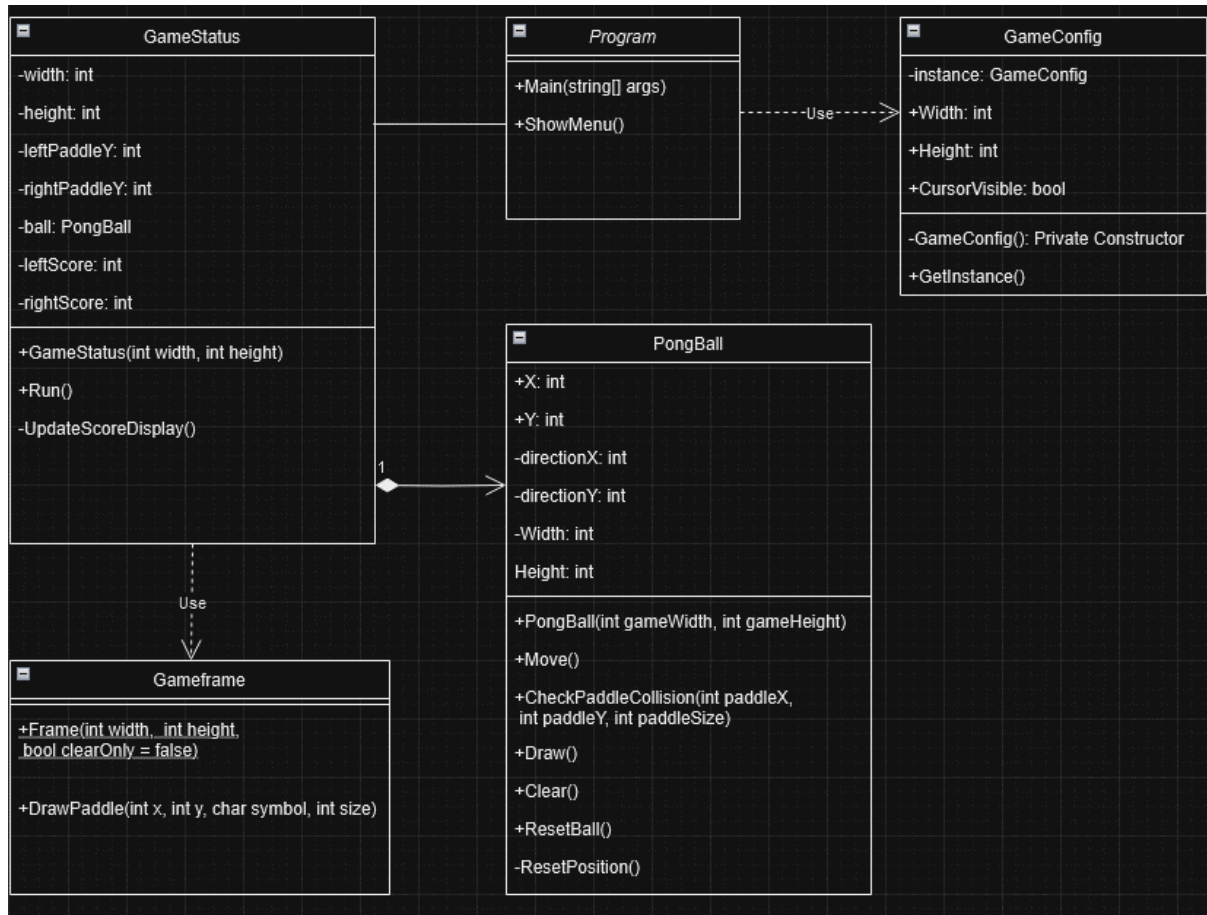
De gebruiker(actor) selecteerd show highcores in de main menu, de programma verifieert deze input en vraagt vervolgens aan de database de benodigde data voor de database. De database stuur deze gegevens terug naar de pong app en de pong app formateert dit data en laat het uiteindelijk zien aan de actor.



UML Klassen Diagram

Klassen Diagram: laat de structuur van een systeem weergeeft door de systeemklassen, hun attributen, operaties (of methoden), en de relaties tussen de klassen te tonen.

Klassediagram: Laat zien welke klassen er zijn in de pong game en hoe deze relateren met elkaar



1. Program

Dit is de hoofdklasse die het begin bevat van de applicatie de main methode heeft methodens als: ShowMenu om de hoofdmenu te zien

2. GameStatus

Deze klasse houdt de status van het spel bij, zoals width, height, leftpaddleY, rightpaddleY en de scores van de speler. Het heeft ook een run methode om de spel te starten en een UpdateScoreDisplay methode die de scores update

3. PongBall

Deze klasse voegt de bal toe in de pong game met de eigenschappen zoals de x en y as van de pong bal, de directies van de pong bal en heeft de volgende methodes:

- `PongBall(int gameWidth, int gameHeight)`: Initialiseert de bal met de afmetingen van het spel.
- `Move()`: Beweegt de bal op het speelveld.
- `CheckPaddleCollision()`: Controleert op botsingen met paddles.
- `Draw()`, `Clear()`: Tekent en wist de bal op het scherm.
- `ResetBall()`, `ResetPosition()`: Reset de bal naar de beginpositie.

4. GameFrame

Deze klasse is verantwoordelijk van het tekenen van de GameFrame en de Paddles en heeft de volgende methodes

- `Frame (int width, int height, bool clearOnly)`: Tekent het frame van het spel.
- `DrawPaddle(int x, int y, char symbol, int size)`: Tekent een paddle op de gegeven locatie.

5. GameConfig

Beheert de configuraties van de instellingen via een singleton desing pattern. De GameConfig heeft de volgende methodes:

- `GameConfig()`: Privé constructor die de configuratie initialiseert.
- `GetInstance()`: Geeft de enige instantie van de configuratie terug.

6. Relaties

- De `GameStatus` klasse maakt gebruik van de `PongBall` klasse om de toestand van de bal te beheren.
- De `Program` klasse gebruikt de `GameConfig` klasse om toegang te krijgen tot spelconfiguraties.
- Het 'gebruikt'-relatie is aangegeven tussen `GameStatus` en `GameFrame` voor het bijwerken en tekenen van de spelinterface.

Solid

Bij het opstellen van de klassen in de applicatie zelf is veel aandacht besteed aan de 5 Solid principles. De 5 richtlijnen zijn als volgt:

1. Single Responsibility Principle (SRP)

Een klasse moet slechts een reden hebben om te veranderen. Dit betekent dat het 1 taak of verantwoordelijkheid moet hebben.

2. Open/Closed Principle (OCP)

Klassen moeten open staan voor uitbreidingen, maar gesloten zijn voor wijzigingen. Je moet functionaliteit kunnen toevoegen zonder bestaande code te veranderen.

3. Liskov Substitution Principle (LSP)

Objecten van een superclass moeten kunnen worden vervangen door objecten van subclasses zonder dat dit de correctheid van de programmeer beïnvloedt.

4. Interface Segregation Principle (ISP)

Geen code moet afhankelijk zijn van interfaces die het niet gebruiken. Grotere interfaces moet je kunnen opsplitsen in kleinere interfaces die specifieke taken kunnen uitvoeren.

5. Dependency Inversion Principle (DIP)

Interfaces moeten een helder omschreven doel hebben, vergelijkbaar met het Single Responsibility Principle. Grote interfaces kunnen leiden tot implementatieverplichtingen voor klassen die niet alle methodes nodig heeft.

Source: <https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/>

Solid in de pong applicatie + (KlassenDiagram):

1. Single Responsibility Principle (SRP)

De "GameStatus" klasse beheert alleen de spelstatus, terwijl de GameFrame zich focust op de weergave.

2. Open/Closed Principle (OCP)

Nieuwe functionaliteiten zoals **PongBall** kunnen worden toegevoegd zonder de bestaande code te wijzigen door een nieuwe subclass te creëren.

3. Liskov Substitution Principle (LSP)

De **PongBall** kan veranderd worden in gederag maar de **GameStatus** klasse verwacht dat alle ballen zich aanpassen aan de gemeenschappelijke interface.

4. Interface Segregation Principle (ISP)

Dit wordt toegepast door de functionaliteiten in de applicatie op te delen in interfaces.

5. Dependecy Inversion Principle (DIP)

De **Program** Klasse is afhankelijk van de abstractie van de **GameConfig** en niet van de details van de implementatie.

DesingSmells

Betekenis:

Desing smells zijn indicatoren binnen de code vermoeden dat er problemen zijn met het software ontwerp. Ze zijn gericht op code die moeilijk is te onderhouden of uit te breiden is.

Pong Applicatie:

Te veel verantwoordelijkheden: Een klasse die bijvoorbeeld meerde rollen heeft in de pong applicatie. Dit gaat tegen de principe van Single Responsibility Principle.

Lange methodes:

Methodes die meer doen waarvoor ze nodig zijn of ontwikkeld zijn.

Overmatige koppeling:

Het vermeiden van afhankelijkheid componenten, dit zorgt ervoor dat de flexibiliteit verminderd word en het moeilijk maakt om componenten te testen of wijzigen

Hoe wij hiermee omgaan om desingsmells te vermijden is:

1. **Refactoren** Code refactoren en apart te zetten in hun eigen sub files dit zorgt voor leesbaarheid en een duidelijke structuur in de applicatie.
2. **Code reviews** Regelmatig code reviews met elkaar te hebben, het kan zijn dat jou medestuden iets over de oog ziet. Dit kan je vermijden om met elkaar de code te reviewen.
3. **Leesbaarheid** Comment de code dat word geschreven en beschrijf wat er gebeurt, zo vermeid je verwaring tussen de developers heen.

DesingPatterns

Desing Patterns

Zijn herbruikbare oplossingen voor veelvoorkomende problemen in softwareontwerp.

Creational Patterns:

Deze patronen zijn gericht op het creëren van objecten op een manier die geschikt is voor de situatie. Het basisconcept is om het creëren van objecten te scheiden van hun gebruik.

Voorbeelden: Singleton, Factory Method, Abstract Factory, Builder, Prototype.

Behavioral Patterns:

Gedrag patronen zijn gericht op efficiënte communicatie en de toewijzing van verantwoordelijkheden tussen objecten. Ze helpen bij het definiëren van hoe objecten met elkaar samenwerken.

Voorbeelden: Observer, Strategy, Command, Iterator, State, Visitor.

Concurrency Patterns

Concurrency patronen richten zich op het ontwerpen van multi-threaded (parallele) applicaties en het omgaan met de uitdagingen die komen kijken bij gelijktijdige uitvoering van componenten.

Voorbeelden: Active Object, Monitor Object, Half-Sync/Half-Async, Leader/Followers.

Structural Patterns

Structurele patronen zijn gericht op het vormgeven of samenstellen van klassen en objecten om grotere structuren te vormen. Ze helpen bij het verzekeren dat als het ene deel van een systeem verandert, het hele systeem niet hoeft te veranderen.

Voorbeelden: Adapter, Composite, Proxy, Flyweight, Decorator, Bridge.

Elk van deze patronen speelt een cruciale rol in het oplossen van specifieke ontwerpproblemen en het verbeteren van de codeflexibiliteit.

Creational Pattern Singleton

De **GameConfig** klasse is ontworpen volgens de singleton patroon, dit houdt in dat er slechts 1 instantie is tijdens de looptijd van de applicatie. De Singleton GameConfig implementatie zorgt ervoor dat de configuratie van het spel consistent is en centraal beheerd wordt door de hele applicatie.

```
// We create a singleton class for the game configuration, this class is used to set the game window size and the cursor visibility
// The reasoning is that we only need one instance of the game configuration, so we use a singleton pattern to ensure that only one instance is created.
public class GameConfig
{
    private static GameConfig instance;
    public int Width { get; private set; } // Width and height properties for the game window size change
    public int Height { get; private set; }
    public bool CursorVisible { get; private set; }

    // Private constructor to prevent instantiation outside this class
    private GameConfig()
    {
        Width = 90;
        Height = 30;
        Console.OutputEncoding = System.Text.Encoding.UTF8;
        Console.SetWindowSize(Width + 1, Height + 1);
        CursorVisible = false;
        Console.CursorVisible = CursorVisible;
    }

    // Public method to get the instance
    public static GameConfig GetInstance()
    {
        if (instance == null)
        {
            instance = new GameConfig();
        }
        return instance;
    }
}
```

(In de comments word uitgelegd wat er gebeurt)

```
static void Main(string[] args)
{
    ShowMenu();
    Console.ReadLine();

    // Gets the singleton instance of GameConfig
    GameConfig config = GameConfig.GetInstance();
}
```

In de Main roepen we de GameConfig op om de instantie te starten.

Behavioral Pattern

Helaas had ik niet genoeg tijd om een Behavioral pattern te verwerken in het project.

Ik heb wel onderzoek gedaan na wat er mogelijk zou zijn voor een Behavioral Pattern in de Pong Applicatie. We zouden bijvoorbeeld een Command Pattern kunnen coderen voor de inputverwerking van de paddles.

Het voordeel hiervan is: Dat de structuur van de inputverwerking verbeterd word door de acties te scheiden. Dit zorgt er ook voor dat we in de toekomst makkelijker is om meerder functionaliteiten toevoegen. Het geeft ook meer overzicht en is makkelijker te onderhouden.

Concurrency Pattern

Concurrency pattern: De pong applicatie maakt gebruik van concurrency om parallelle verwerkingen mogelijk te maken. Dit betekent dat het programma meerdere processen tegelijkertijd kan uitvoeren.

```
//Concurrency pattern, thread per object
Thread gameThread = new Thread(new ThreadStart(gameStatus.Run));
gameThread.Start();
```

De pong applicatie initialiseert een aparte thread voor de game loop door een thread object te creëren.

Met de **start** methode van de **thread klasse** worden de uitvoeringen van de **run methode** die zich bevindt in de **GameStatus** klasse als een aparte proces gestart.

Het voordeel hiervan is dat we processen van elkaar kunnen scheiden en de game meer responsief hierdoor kunnen maken.

Structural Pattern

Het zelfde geldt voor Structural pattern. Helaas had ik niet genoeg tijd om deze te verwerken in het project. Ik heb wel onderzoek gedaan na wat er mogelijk zou zijn voor een structural pattern in de Pong Applicatie maar had niet genoeg tijd om dit erin te verwerken en documenteren.

Ik zat te denken aan een Decorator pattern, Dit zou helpen om klassen dynamisch uit te breiden met extra functionaliteit. Het voordeel hiervan is dat ik klassen zou kunnen uitbreiden zonder veranderingen te maken aan de onderliggende code. Decorators zouden tijdens runtime toegevoegd of verwijderd kunnen worden waardoor je functionaliteiten kunt combineren en wijzigen.

We zouden bijvoorbeeld geluid kunnen laten afspelen met de Beep() methode als er gescoord wordt. Via de decorator pattern zouden we dit kunnen doen zonder de basisimplementatie van de klasse te wijzigen. Dit zorgt voor schone leesbare code.

Sources

<https://designwizard.com/blog/colour-combination/> **Voor de kleur combinatie**

<https://imagecolorpicker.com/> **Hex + RGB Picker**

<https://www.lucidchart.com/pages/nl/wat-is-unified-modeling-language> **Voor de uitleg wat UML is**

<https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/> **Uitleg van wat Solid is**

https://en.wikipedia.org/wiki/Design_smell **Uitleg van Desing smells**

https://www.wikiwand.com/en/Class_diagram **Class Diagram: Pijltjes (relations).**

https://www.wikiwand.com/en/Component_diagram **Component diagram**

Naast dese sources heb ik ook de docenten persoonlijk benaderd voor feedback en de powerpoints online bestudeerd.

Nawoord

Jammer genoeg had ik niet genoeg tijd om de Behavioral en Structural pattern te kunnen verwerken in mijn code en te documenteren.

Ik heb veel meer kennis kunnen opdoen van Scrum en vooral de scrumboard, epics, userstories. Ook weet ik nu hoe ik wireframes goed kan maken en kan documenteren.

Voor UML heb ik ook veel geleerd hoe het allemaal verder in elkaar zit qua diagrammen waarom we ze maken en hoe ik ze kan implementeren om overzicht te geven van mijn applicaties.

Verder heb ik persoonlijk extreem veel geleerd in beide vakken en is er ook toch wel wat passie ontstaan voor C# en het behandelen van Design Patterns.