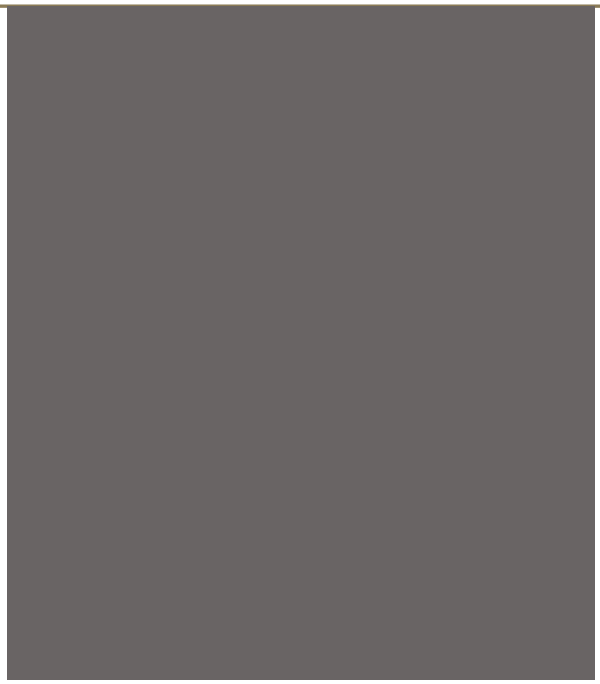


SKRRRT

Software architectuur document

Santos Marques,Pedro P. DosGroep C

Inhoud

H1 Inleiding

Doel van document

In dit document wordt er een overzicht gegeven van de architectuur aspecten van SKRRRT. Dit houdt in de componenten verdeling, networking aspecten, deployment en persistentie

Deze document zal groeien naar mate dat onze team het project verder ontwikkelt. Dit betekent dat de diagrammen nog aangepast zullen worden om de eisen van de opdrachtgever voor elke iteratie te kunnen voldoen.

Library

De diagrammen worden in Visual Paradigm gemaakt en zijn compleet met uitleg. De niet-standaard manier waarop wij onze project hebben opgesteld (heftig gebruik van library specifiek structuur i.p.v. standard Java practices in sommige gevallen) heeft te maken met het feit dat onze project opgebouwd is met basis op de libGDX library.

Deze library vereist een specifieke opzet van klassen en interfaces dat - vergeleken met het gebruik van Java-standaard libraries zoals JavaFX of Swing - een beetje overweldigend of onleesbaar kan zijn voor mensen die niet bekend zijn met libGDX. Alle basics en niet-standaard practices die op onze project kunnen voorkomen worden in ons Onderzoek Document in detail uitgelegd. Dit betekent dat alles wat wij hebben geprogrammeerd een extensie is van wat op ons onderzoek document is geschreven.

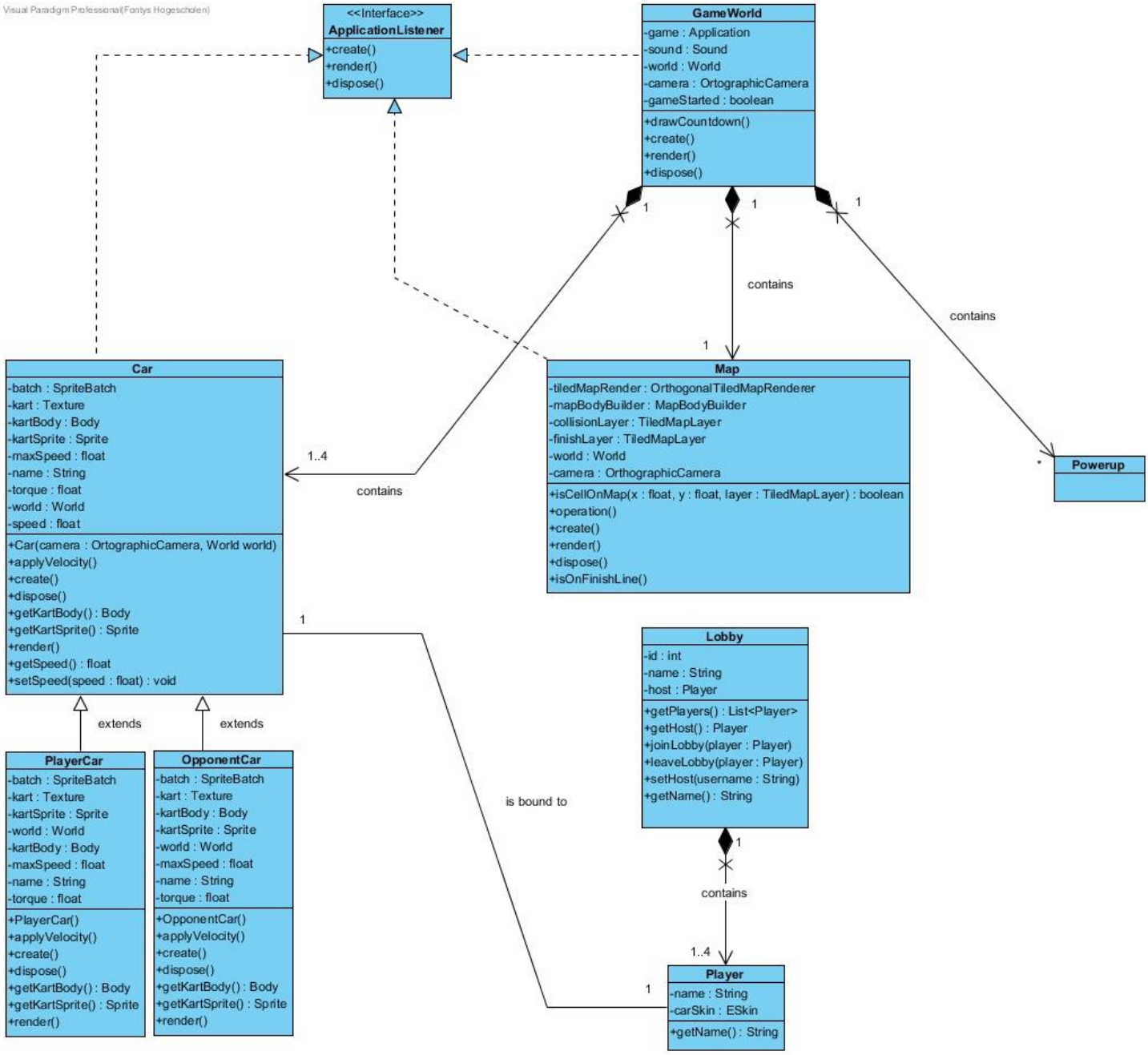
Context

Het doel van ons project is het ontwikkelen van een multiplayer racing game in Java. Het is de bedoeling dat deze racing game in de vorm van gedistribueerde software ontwikkeld wordt. In SKRRRT, 2 tot 4 spelers zullen elkaar racen en vernietigen om als 1st te finishen op de racingbaan.

In iteratie 2 heeft onze team zich gefocust op het ontwerpen van een sterke networking foundation waarop wij makkelijk multiplayer games kunnen hosten

H2 Domeinmodel

Klassendiagram

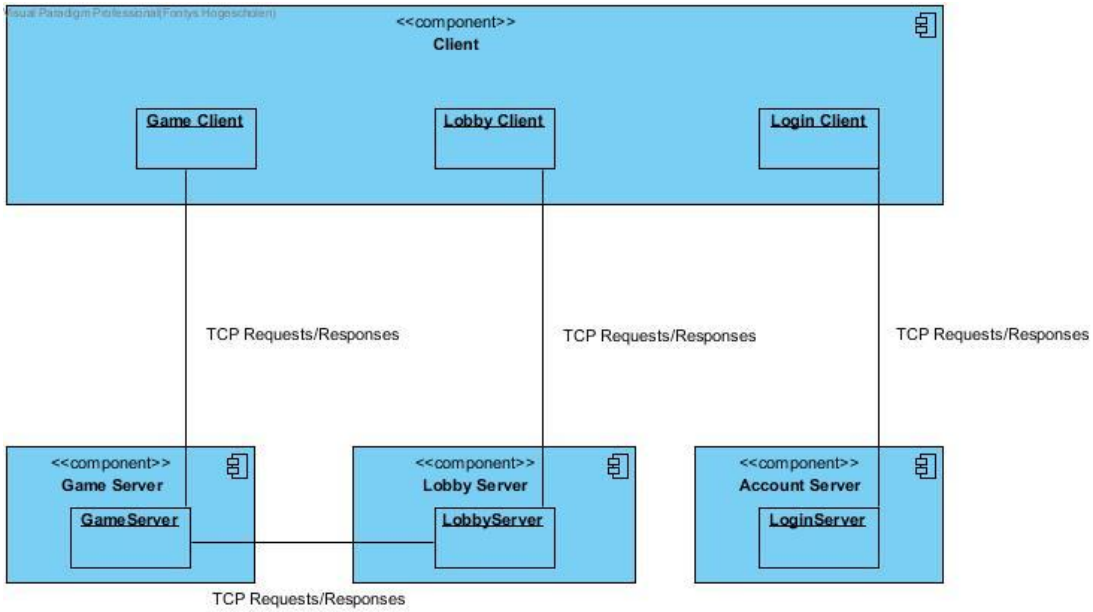


Figuur 1: Klassendiagram van het domein.

In figuur 1 is het klassendiagram van het domein afgebeeld. Hierin worden de volgende klassen onderscheiden:

- GameWorld: Representeert de hoofdklasse van het spel. Hierin worden de klassen Car, Map en PowerUp opgenomen en gebruikt. Daarnaast implementeert deze klasse de interface genaamd ApplicationListener om de 'create', 'render' en 'dispose' methodes over te nemen
- ApplicationListener: Representeert een interface. De 'create' methode wordt gebruikt om bij opstart bijvoorbeeld iets te laten initialiseren. De 'render' methode wordt gebruikt om herhaaldelijk iets te laten doen. Als laatste wordt de 'dispose' methode gebruikt om bepaalde objecten, variabelen te verwijderen i.v.m. geheugencapaciteit.
- PowerUp: Representeert een power up van een auto. Deze klasse is nog in opbouw en daarom alleen nog gekoppeld aan GameWorld, maar nog niet aan Car.
- Map: Representeert een map van het spel. Herin wordt de map van het spel getekend. Daarnaast wordt er ook gecontroleerd of een auto de eindstreep heeft gepasseerd. Ook implementeert deze klase de interface genaamd ApplicationListener om de 'create', 'render' en 'dispose' methodes over te nemen.
- Car: Representeert een auto van het spel. Hierin wordt de auto getekend. Ook implementeert deze klase de interface genaamd ApplicationListener om de 'create', 'render' en 'dispose' methodes over te nemen.
- PlayerCar: Representeert de auto van de huidige gebruiker die is ingelogd.
- OpponentCar: Representeert een auto van een tegenstander.
- Player: Representeert een gebruiker van het spel. Hierin wordt benodigde informatie opgeslagen van de gebruiker. Zoals naam en 'skin' van de auto. De gebruiker zal een 'skin' voor het spel kunnen selecteren in het menu.
- Lobby: Representeert een lobby van het spel. Hierin worden alle 'players' opgeslagen. Deze klasse bevat ook een gebruiker die de lobby heeft aangemaakt ('host'). Alleen deze gebruiker kan de naam en map wijzigen.

H3 Opdeling in Componenten



Figuur 2: Componenten diagram.

Componentendiagram

Het componentendiagram in figuur 2 laat de 4 componenten van SKRRRT zien met de klassen en interfaces die ze aanbieden/ontvangen. De componenten zijn:

- Client
  - Representeert de software applicatie die de gebruiker gebruikt.
- Game Server
  - Representeert de software applicatie die de voortgang van de game berekent en bijhoud.
- Lobby Server
  - Representeert de software applicatie die bijhoud welke gebruikers in welke lobby zit. Gebruikers kunnen zich ook aanmelden en afmelden voor de lobby.
- Account Server
  - Representeert de software applicatie die het inloggen verzorgt. Deze applicatie kijkt of de gebruiker bestaat en of deze het bijpassende wachtwoord heeft.

Koppeling tussen componenten

Alle servers worden gekoppeld aan de bijbehorende Client klasse binnen de Client-applicatie. Op deze manier worden verschillende taken door verschillende servers verwerkt en ontstaat er het Single Responsibility Principle (SRP). Op deze manier heeft elke server zijn eigen verantwoordelijkheid. De klassen in de Client worden dan aangesproken op het moment dat deze nodig zijn.

Synchronisatie tussen componenten

Synchronisatie is op twee plekken van groot belang.

- Client-Lobby
  - Hier is synchronisatie van belang omdat het belangrijk is voor de gebruiker om te kunnen zien welke lobby's beschikbaar zijn. Ook als een andere gebruiker/Client zich bij een lobby aanmeld/afmeld dan moet dit ook zichtbaar worden voor de gebruikers. Ook bij het creëren en verwijderen van lobby's moet dit meteen aangepast worden anders is een gecreëerde lobby niet zichtbaar en is het niet mogelijk bij een verwijderde lobby aan te melden.
- Client-Game
  - Hier is synchronisatie belangrijk omdat de gebruiker tijdens het spelen van het spel ook de andere gebruikers wil kunnen zien verplaatsen. Dit is misschien wel het meest essentiële onderdeel van een multiplayer game. Zonder synchronisatie tussen deze componenten en de verschillende Clients is het eigenlijk geen multiplayer game meer en is de game meer een race tegen de klok.

Services per component

Per component worden de volgende services aangeboden:

- Login Client
  - LoginRequest
- Lobby Client
  - CreateLobbyRequest
- Game Client
  - GameStartRequest
- Game Server
  - GameStartResponse
- Lobby Client
  - CreateLobbyResponse
- Login Server
  - LoginResponse

Communicatie tussen de verschillende clients/servers gaat in de vorm van het uitwisselen van TCP pakketten. Voor meer informatie hierover zie de hoofdstuk Communicatie.

Allocatie van objecten

Hier worden beschreven waar welke instanties van welke objecten zijn gealloceerd.

- Game Server: Instantie van GameServer.
- Lobby Server: Instantie van LobbyServer.
- Account Server: Instantie van LoginServer.
- Client: Instanties van GameClient, LobbyClient en LoginClient;

Packagestructuur

- Gameplay: In deze package zit alles met het besturen van de auto.
- Map: In deze package zit alles wat te maken heeft met de map.
- Networking: In deze package zit alles wat te maken heeft met networking.

- De verschillende Clients die onderdeel zijn van de hoofd Client.
- De verschillende Servers die de componenten voorstellen.
- Alle network requests/responses
  - MenuScreen: alle screens voor de Client van de gebruiker.

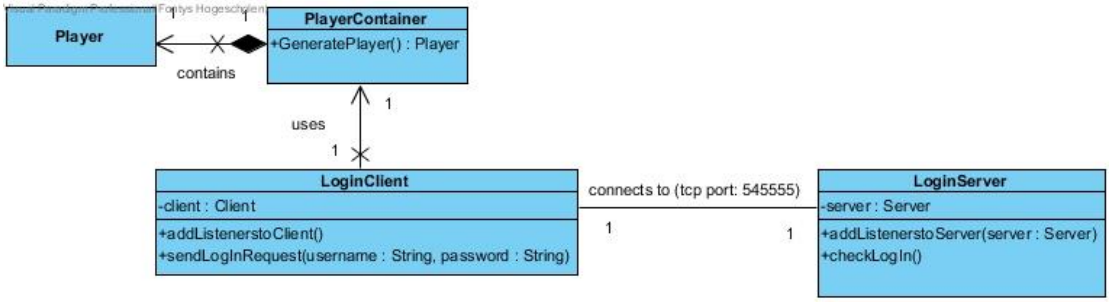
H4 Communicatie binnen SKRRRT

Communicatie Library

Communicatie tussen de componenten van SKRRRT wordt gerealiseerd door het gebruik van uitwisseling van pakketten.

Om dit te coderen er wordt gebruik gemaakt van de Kryonet networking library (https://github.com/EsotericSoftware/kryonet). Er werd hiervoor gekozen omdat Kryonet versimpelt het opstellen van pakketten-communicatie door interfaces en serialisatie automatisch te doen, en maakt gebruik van methoden die goed te combineren zijn met het observable pattern.

Communicatie Client – Account Server



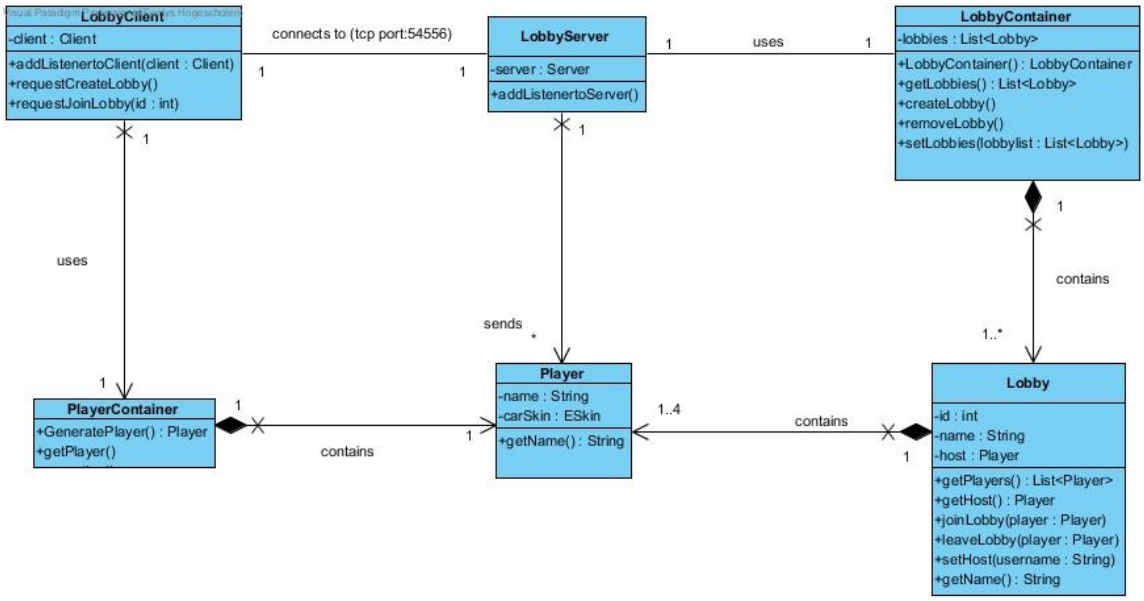
Figuur .. : Communicatie Client-Account server

De communicatie tussen de Client en de Account server gaat via het uitwisselen van requests en responses.

Het proces bij het aanmelden gaat als volgt:

- Client stuurt een request met de ingevulde gebruikersnaam en wachtwoord naar de Account server.
- De Account server checkt met de database of de inloggegevens correct zijn. En stuurt een response boolean terug. Is de gebruiker bekend en heeft hij het juiste wachtwoord dan wordt er een true teruggestuurd in de andere gevallen is het een false.
- In het geval van een response-true maakt de LoginClient via de PlayerContainer een nieuwe Player aan.

Communicatie Client – Lobby Server



Figuur ..: Communicatie Client-Lobby server

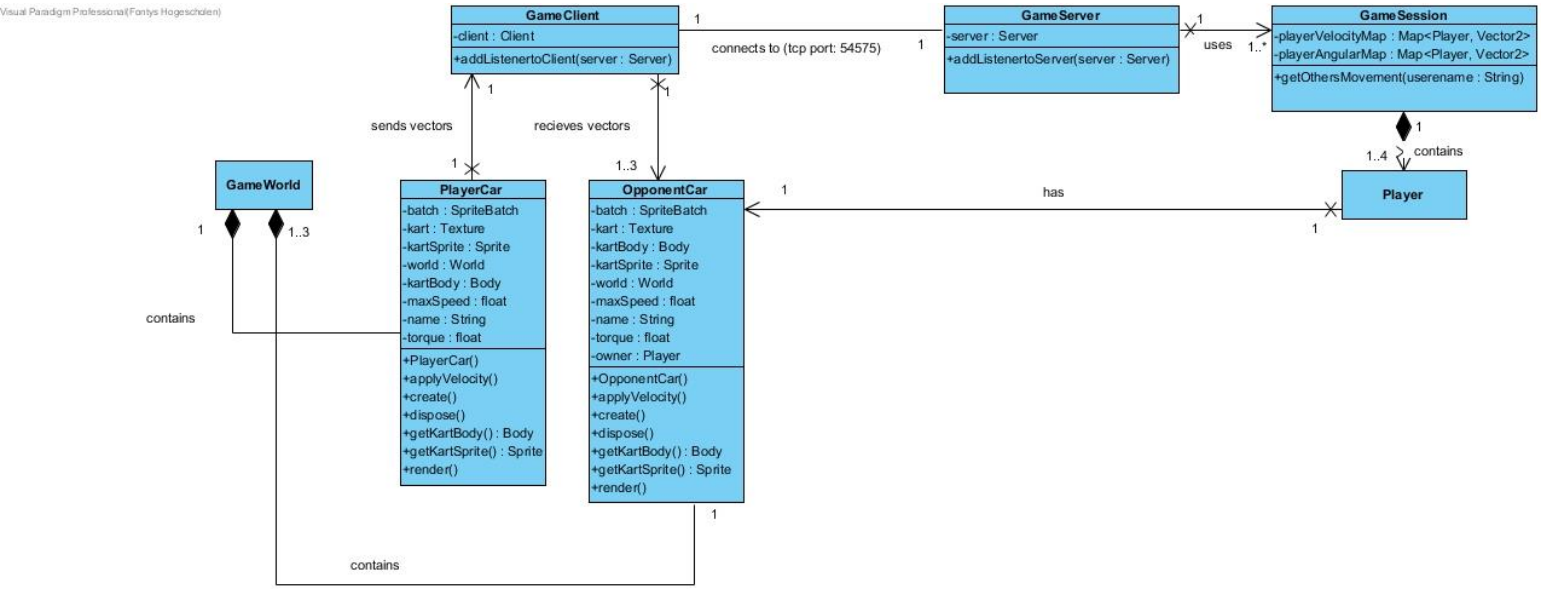
Communicatie tussen de Client en Lobby server component wordt gerealiseerd door middel van uitwisseling van requests en responses. De requests en responses bevatten attributen die over het netwerk verstuurd worden. In dit geval worden er Players en Lobby-ids verstuurd.

Er vindt communicatie plaats op de volgende momenten:

- Het aanmaken van een nieuwe lobby
  - Client stuurt een request om een nieuwe lobby aan te maken en geef daarbij de naam van de lobby op.
  - De Lobby server maakt een nieuwe lobby met de meegegeven naam aan en plaatst deze in de container.
  - De Lobby server stuurt de aangepaste lijst met lobby's naar alle verbonden Clients.
  - De Client verwerkt deze lijst en laat de nieuwe lobby's zien inclusief de nieuwe lobby.
- Het aanmelden voor een bestaande lobby
  - Client stuurt een request om voor een lobby aan te melden. Hierbij wordt de Player en de lobby naam meegegeven.
  - De Lobby server zoekt in de lobby container de juiste lobby en voegt de Player toe aan de juiste lobby.
  - De Lobby server stuurt, als response, de lijst met spelers van de aangepaste lobby naar alle clients waarvan de spelers is die lobby zitten.
  - Client van alle gebruikers in die lobby laden de nieuwe lijst in.
- Afmelden bij een lobby
  - Client stuurt een request de huidige lobby te willen verlaten. Hierbij wordt een lobby id en de Player meegegeven.
  - De Lobby server verwijdert de Player van de juiste lobby.
  - De Lobby server stuurt, als response, de lijst met spelers van de aangepaste lobby naar alle clients waarvan de spelers in die lobby zitten.
  - Client van alle gebruiker in die lobby laden de nieuwe lijst in.
- De host start het spel

- Op het moment dat de host het spel start, stuurt de Client een startrequest naar de Lobby server.
- De Lobby server stuurt vervolgens een request naar de Game server dat het spel kan beginnen, in die request worden de spelers in de lobby meegegeven.
- De Game Server handelt dit verder af.

Communicatie Client – Game Server

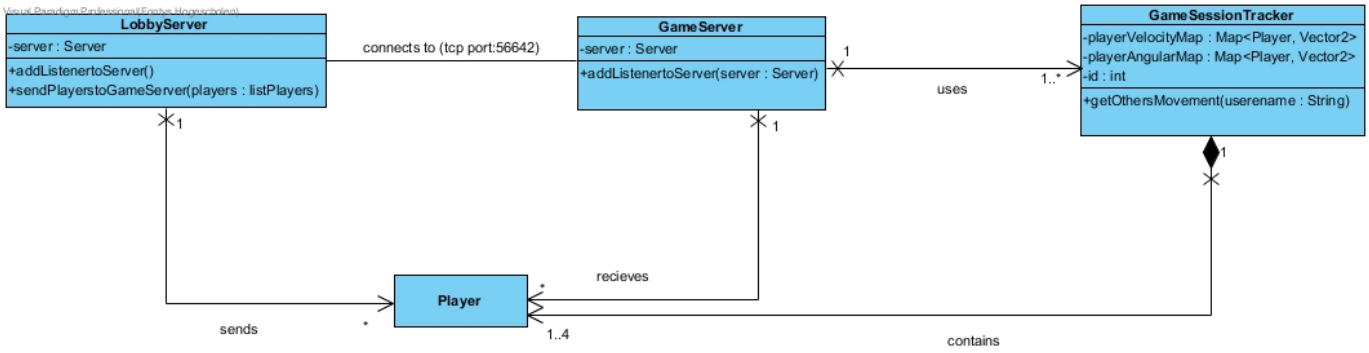


Figuur ..: Communicatie Client-Lobby server

Communicatie tussen de Client en Game Server component wordt gerealiseerd door middel van uitwisseling van requests en responses. De requests en responses bevatten attributen die over het netwerk verstuurd worden. In dit geval worden er Vector2 en strings verstuurd.Communicatie tussen deze twee componenten vindt plaats op de volgende momenten:

- Bij het beginnen van een game: Client stuurt een readyGame packet naar server om de GameSession te laten weten dat de speler klaar is om te spelen.
- Server stuurt een gameStart package naar alle verbonden clients om de game bij de clients te starten.
- Zodra de game is begonnen, er worden continu movementUpdateRequest packets door de client verstuurd en movementUpdateResponses ontvangen
  - De movementUpdateRequest packets bevatten de snelheid en de hoeksnelheid van de auto van de speler (local dus). Die worden tijdelijk in GameSessionopgeslagen totdat een nieuwe movmentUpdateRequest wordt verstuurd.
  - De movementUpdateResponse packets bevatten de snelheid en hoeksnelheid van de autos van de andere spelers verbonden aan de GameSession. De vectoren worden gebruikt door de OpponentCar instanties om de andere auto's te laten bewegen.
- Als de finishline is bereikt door iemand, wordt een finisshLineReachedRequest door de client naar de server verstuurd, en vervolgens een finishLineNotice door de server naar alle spelers in de GameSession. Dit triggert een waarschuwing aan alle players dat laat weten dat iemand de finish lijn heeft bereikt.
- Als alle spelers de finishline hebben bereikt kunnen de spelers de game verlaten en wordt de GameSession vernieuwd.

Communicatie Lobby Server – Game Server



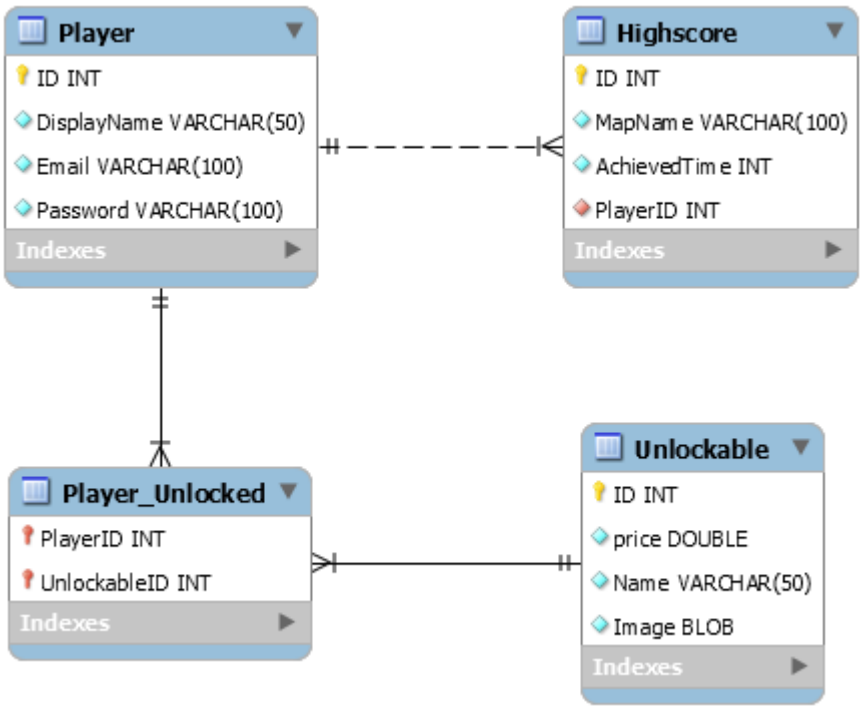
Figuur X: Lobby Server – Game Server communicatie

Communicatie tussen de Lobby Server en Game Server component wordt gerealiseerd door middel van uitwisseling van requests en responses. De requests en responses bevatten attributen die over het netwerk verstuurd worden. In dit geval worden er Vector2 en strings verstuurd.

Communicatie tussen deze twee componenten vindt plaats als een lobby leader (een speler) besluit om een game te starten. Dan worden alle spelers die in de lobby zijn als objecten verstuurd naar de GameServer door de LobbyServer klasse. Daarna wordt de lobby verwijderd.

De players worden opgenomen in een GameSession, waar hun snelheid data wordt opgeslagen om later tussen alle players gedeeld te worden. De werking hiervan wordt in Game Server detailed design verder beschreven (blz. 17).

H5 Persistentie per component



Figuur …: Database ontwerp

Persistentie binnen SKRRRT

Om de login gegevens, highscores en vrijgespeelde onderdelen bij te houden wordt er gebruikt gemaakt van een MySQL database.

Er zijn drie entiteiten:

- Player: representeert een gebruiker.
  - Displayname: naam van de gebruiker. Deze is uniek.
  - Password: wachtwoord om mee in te loggen.
  - Email: emailadres om met de gebruiker te informeren.
- Highscore: representeert de highscores.
  - MapName: naam van de map waar de highscore is gereden.
  - AchievedTime: tijd van de behaalde highscore.
  - PlayerID: ID van de speler die de des betreffende highscore heeft gereden.
- Unlockable: representeert de vrij te spelen/te kopen onderdelen.
  - Price: prijs van het onderdeel.
  - Name: naam van het onderdeel.
  - Image: afbeelding van het onderdeel.

De entiteiten Player en Unlockable zijn aan elkaar gekoppeld met behulp van een koppeltabel. Op deze manier zijn meerdere onderdelen vrij te spelen per speler en kunnen meerdere spelers hetzelfde onderdeel vrijspelen.

Data opslag tijdens gebruik van SKRRRT

Momenten dat de database wordt aangesproken of dat er data wordt opgeslagen.

- Inloggen: na het opstarten van de client zal de gebruiker moeten inloggen. Op het moment dat de gebruiker zijn naam en wachtwoord heeft ingevuld en op de login-knop heeft gedrukt, zal de database gevraagd worden of deze gebruiker bestaat en of hij het juiste wachtwoord heeft ingevuld.
- Highscore: aan het einde van de race wordt de database gevraagd wat de huidige snelste tijd van die gebruiker is op die map. Mocht de nieuwe tijd sneller zijn dan de oude tijd, wordt de nieuwe tijd opgeslagen.
- Unlockable: Op het moment dat een gebruiker een nieuw onderdeel vrijspeelt wordt de database hiervan op de hoogte gebracht en zal deze het onderdeel aan de gebruiker koppelen. Zodat de gebruiker de volgende keer hij inlogt nog steeds deze onderdelen vrijgespeeld heeft.

H6 Detailed design per component

Client

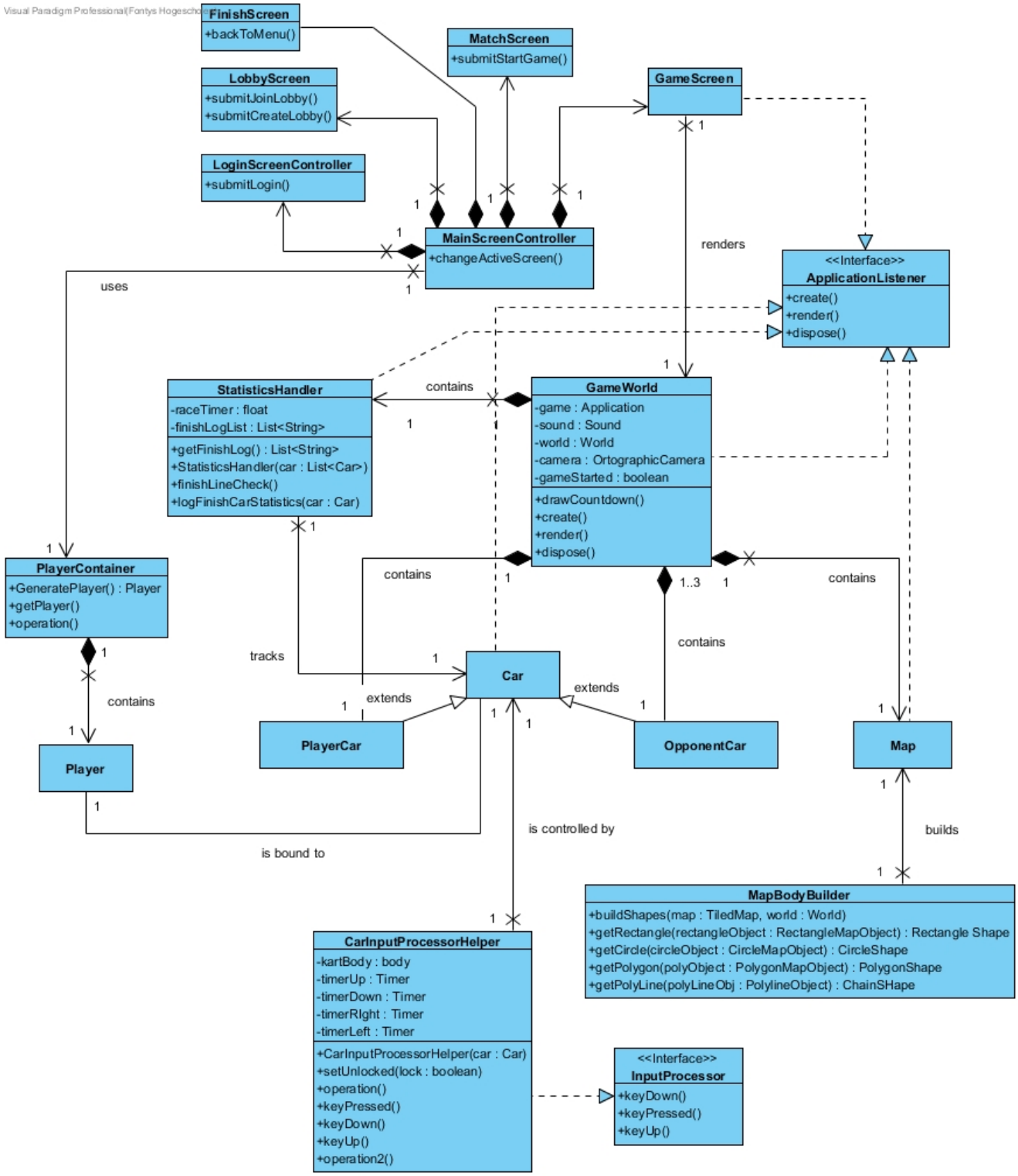


Figure 1

De klassendiagram in figuur 9 is de representatief voor de klassen van het Client component. De Client heeft als functies:

- Als een entrypoint voor de applicatie te dienen;
- Een visuele interface aan de gebruiker aanbieden zodat de hij de menu's en de game zelf kan bedienen door gebruik van mouse en keyboard input;
- Communicatie met de verschillende servers uitvoeren om gameplay met andere spelers mogelijk te maken;

De Client bestaat uit drie lagen:

- Objectenlaag:

- Bestaat uit domeinobjecten en betreffende containers die gebruikt worden in de game. Dat houdt in de Player, de Car, en Map.
- Deze worden in GameWorld geplaatst. De GameWorld wordt gebruikt als een container voor alle game objecten. De PlayerContainer wordt gebruikt als een abstractieklasse om de de player op te slaan tijdens gebruik van de applicatie.
- Alle objecten die een realisatie zijn van ApplicationListener zijn gameobjecten. Dit houdt in dat elke frame worden er methoden uitgevoerd door deze objecten tijdens de game loop. Dit wordt verder beschreven in de SKRRRT Onderzoeksdocument.

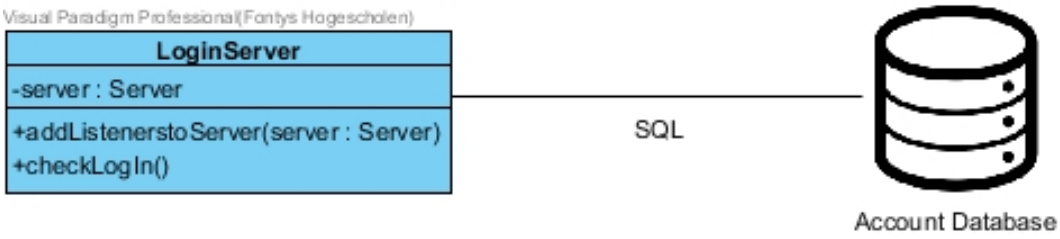
- Communicatielaag:

- Bestaat uit alle klasses die direct met servers communiceren. Elke klasse hier maakt gebruik van een een kryonet Client, die zich verbindt (afhankelijk van de rol) aan een bepaalde server. Daardoor vindt communicatie plaats door middel van uitwisseling van pakketten. Dit staat beschreven in de Communicatie hoofdstuk per casus.

- Presentatielaag:

- Bestaat uit een MainScreenController, en alle Screen klassen. Deze klassen worden gebruikt om informatie in de vorm van menu's en de game aan de gebruiker te laten zien.
- De naam van elke controller geeft aan welke screen het beheert.
  - De login screen toont het inlogscherm in.
  - De lobby screen toont alle lobbies die op de server actief zijn en opties om een lobby te starten of joinen.
  - De match screen toont alle players binnen je lobby, en opties om de game te starten.
  - De game screen toont de game zelf.
  - De finish screen toont hoe alle players het hebben gedaan qua tijd een een game en geeft de mogelijkheid om terug naar de main menu te gaan.
  - Het wisselen van actieve screen wordt in MainScreenController uitgevoerd.

Account Server



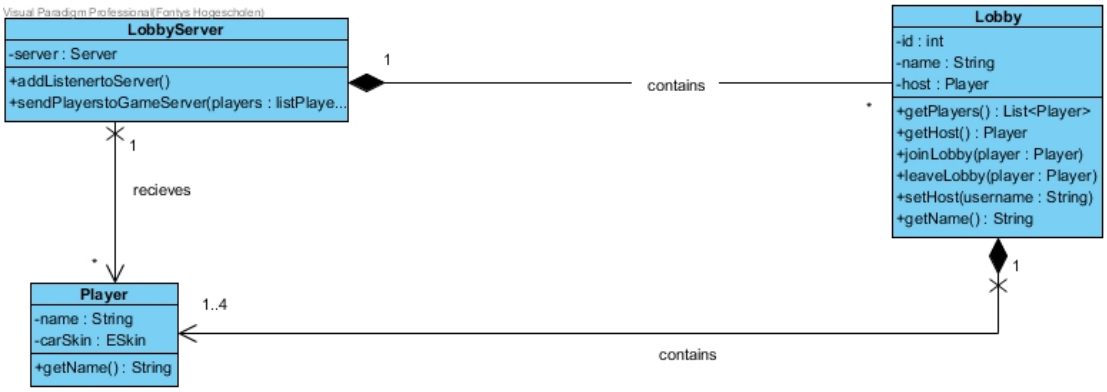
De klassendiagram in figuur 9 is de representatief voor de klassen van het Account Server component.

Het Account Server component heeft als doel de communicatie tussen de Client component en de database beheren. Dit houdt in het maken van parameterized queries naar een database server en het bijhouden van login sessies.

Zoals elke server component in SKRRRT, de LoginServer klasse bestaat uit een Kryonet Server object, die alle networking beheert door middel van een combinatie van het beschikbaar stellen van de server op een netwerk en event listening (events bij een nieuwe verbinding, disconnect, etc.).

Door middel van het uitvoeren van checkLogin worden de gekregen parameters (username en wachtwoord) in een SELECT SQL query gezet en uitgevoerd. Als die query resultaten returnt geeft check login een boolean als return waarde die naar de client gestuurd wordt. De client kan dan inloggen als de boolean waar is.

Lobby Server



De klassendiagram in figuur 9 is de representatief voor de klassen van het Lobby Server component.

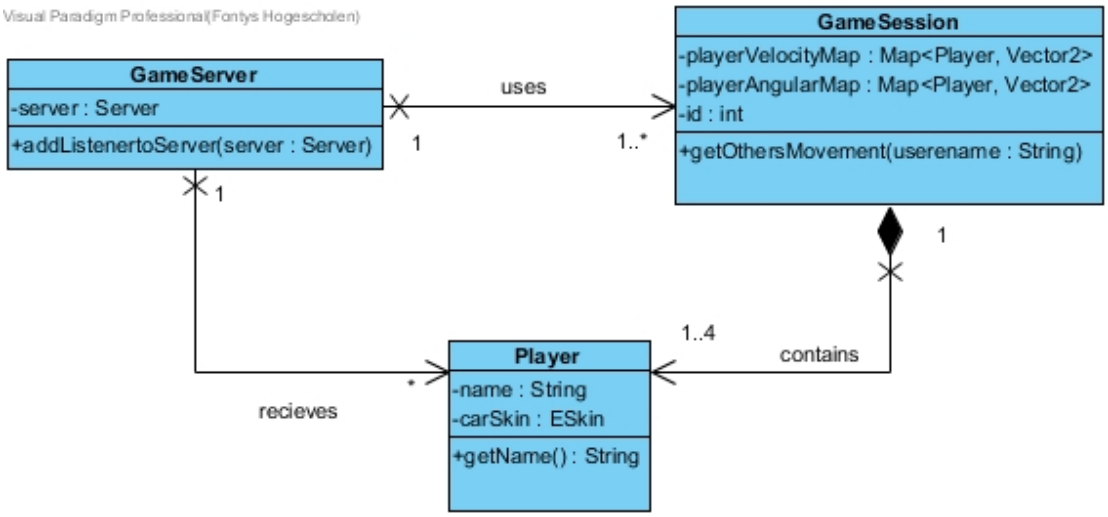
Zoals elke server component in SKRRRT, de LobbyServer klasse bestaat uit een Kryonet Server object, die alle networking beheert door middel van een combinatie van het beschikbaar stellen van de server op een netwerk en event listening (events bij een nieuwe verbinding, disconnect, etc.).

De Lobby Server heeft als functie het aanmaken van lobbies en het beheren van de interacties tussen Player en Lobby, zoals lobbies joinen, spel starten, spelers uit een lobby gooien, etc.

De communicatie details worden verder beschreven in de hoofdstuk Communicatie.

Een LobbyServer bevat meerdere lobbies. Een lobby kan maximaal 4 spelers bevatten. Een lobby kan gecreëerd worden door een speler, de host. De host kan mensen uit de lobby gooien en de game laten starten. Nadat een game is gestart, wordt de lobby verwijderd.

Game Server



De klassendiagram in figuur x is de representatief voor de klassen van het Game Server component.

Het Game Server component heeft als doel een game sessie synchroniseren tussen alle spelers. Dit houdt in het continu sturen van hoe snel elke auto in het spel zich beweegt en draait. Deze gegevens worden in elke client opgenomen om alle autos te bewegen volgens de ingevoerde input van elke speler. De game server houdt ook bij welke spelers door de finish line zijn gepasseerd, en geeft deze informatie aan alle spelers.

Zoals elke server component in SKRRRT, de GameServer klasse bestaat uit een Kryonet Server object, die alle networking beheert door middel van een combinatie van het beschikbaar stellen van de server op een netwerk en event listening (events bij een nieuwe verbinding, disconnect, etc.).

GameSessions houden de beweging van elke player in een game sessie bij. De beweging van elke player wordt vertaald naar twee vectoren, die continu naar alle players wordt verstuurd. Als een spel tot einde komt, wordt de GameSession vernieuwd. (Alle players eruit gegooid). Als een player tijdens een spel zijn verbinding verliest wordt hij automatisch verwijderd van de GameSession.

H7 Realisatie niet-functionele eisen

Betrouwbaarheid

- Game moet runnen zonder hinderende latency (50 ms)
  - Zie Performance.
- Inputdelay is minder dan 5 ms
  - De inputmethode zijn licht en kunnen daarom snel werken.

Performance

Om de performance van het programma hoog te houden is er gekozen om te communiceren via Kryonet ipv Remote Method Invocation (RMI). RMI werkt niet snel genoeg om een gedistribueerde game te runnen en daarom is er gekozen om een andere manier van communicatie te zoeken. Dit is Kryonet geworden. Dit is een communicatie library die een gedistribueerde game kan ondersteunen. Door deze keuze wordt de gewenste performance gewaarborgd.

Beveiliging

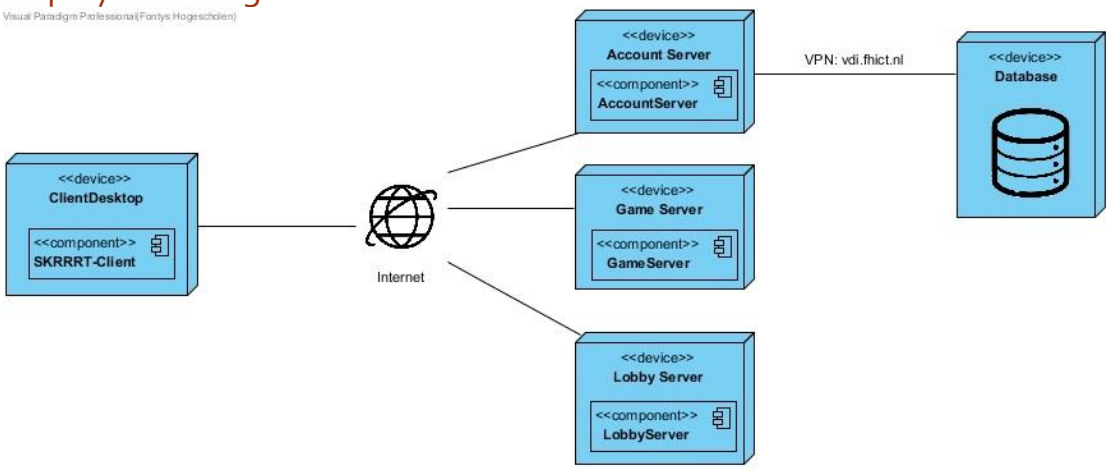
Query's worden aan de hand van parameters gemaakt, op deze manier is er geen SQL injectie mogelijk. Daarnaast wordt er gebruik gemaakt van sessies. Op deze manier wordt ervoor gezorgd dat er een gebruiker zich eerst moet valideren door middel van inloggen. Zo kan niet iedereen zomaar gebruik maken van de game.

Onderhoud

Het systeem is modulair opgebouwd, doordat het is opgebouwd in verschillende componenten. Alle componenten zijn te vervangen.

H8 Deployment

Deploymentdiagram



Het deployment diagram bestaat uit de volgende onderdelen:

- ClientDesktop: de computer waar de SKRRRTClient.jar executable aanwezig is.
- Account Server: een server waar de SKRRRTAccountServer.jar executable aanwezig is.
- Game Server: een server waar de SKRRRTGameServer.jar executable aanwezig is.
- Lobby Server: een server waar de SKRRRTLobbyServer.jar executable aanwezig is.
- Database: een server op het vdi.fhict.nl netwerk waar de database draait.

De koppeling tussen de verschillende servers verloopt allemaal via het internet, behalve de koppeling tussen de Account Server en de database. Hiervoor is een VPN-connectie nodig met het netwerk vdi.fhict.nl. Dit omdat de database gehost wordt op dat netwerk.