Technische Universiteit
Eindhoven
University of Technology

Project 2IM24 - 2008-2009 (groep 2)

Project Etaris

Software Requirements Document

Haan de, S
Ng, HS
Penterman, AGJ
Roulaux, R
Vliembergen van, RWL
Hilkens, RGM

24 november 2008
v 0.1.9

Gr2 Dev

Inhoudsopgave

1 Inleiding

1.1 Doel

Dit SRD geeft een vertaling van de specifieke eisen, beschreven in hoofdstuk 3 van het URD, naar software eisen. In tegenstelling tot de gebruikerseisen, die de wensen van de klant beschrijven, representeren de software-eisen het gedrag van het systeem zoals van buitenaf gezien.

De software-eisen beschrijven wat het *Etaris* moet doen, maar niet hoe dit gedaan moet worden. De software-eisen zijn dus onafhankelijk van de implementatie.

Deze eisen zijn gemodelleerd in een logisch model. Uit dit model moet blijken hoe *Etaris* in elkaar zit en hoe het zich gedraagt.

1.2 Definities

SRD: Software Requirements Document
SR: Software Requirement
URD: User Requirements Document
Actieve speelstuk: Het speelstuk dat als laatste het scherm binnen is gekomen.
Bewegingsrichting: Naar beneden in het bovenste speelveld, naar boven in het onderste speelveld.
Blok: Een eenheid om aan te geven hoe elk speelstuk is opgebouwd en om de breedte en hoogte van de speelvelden aan te geven.
Configuratiebestand: Een tekstbestand waarin de grootte van het speelveld en de vormen van de speelstukken in worden opgeslagen.
Dummy topscore: Een topscore die niet door een speler behaald is maar bestaat om de lijst te vullen. De scores bij deze topscores zijn allemaal 0. De namen bij deze dummy topscores zijn allemaal verschillend.
Game-over: Indien na het verwijderen van alle volle rijen het actieve speelstuk niet meer in zijn bewegingsrichting kan vallen, en niet meer binnen de randen van het bijbehorende speelveld past, is het Game-over.
Hoofdmenu: Het openingsscherm van *Etaris*
Kleur: Een van de kleuren rood, geel en blauw.
Niveau: Moeilijkheidsgraad van het spel, een hoger niveau levert meer punten op maar stukken vallen sneller. Niveaus worden aangegeven met een cijfer met 1 als makkelijkste niveau en 10 als moeilijkste.
Optiemenu: Het scherm waar de gebruiker kan kiezen om een ander configuratiebestand in te laden, het spel naar de originele instellingen te brengen, de topscorelijst te wissen en om terug te gaan naar het hoofdmenu.
Pauzemenu: Het menu dat verschijnt als het spel wordt gepauzeerd.
Rij: Een horizontale strook, over de volle breedte van een speelveld, van 1 blok hoog.
Speelveld: Een veld waar speelstukken in vallen/stijgen. De breedte en hoogte van het veld worden gemeten in blokken.
Snelheid: Het aantal rijen dat een speelstuk per seconde omlaag valt.
Speed-up rate: De versnelling waarmee de snelheid van de speelstukken continu toeneemt, uitgedrukt in rijen per minuut.
Speelscherm: Scherm bestaande uit 2 speelvelden en twee schermen die laten zien wat het volgende speelstuk wordt dat gaat vallen in het bijbehorende speelscherm.
Speelstuk: Een verzameling verbonden blokken die in het spel wordt gebruikt om rijen op te vullen.
Spel: Een spel bestaat uit een venster met 2 speelvelden waarin de speelstukken in de bewegingsrichting bewegen. Het doel van spel is zoveel mogelijk punten halen.
Speler: De gebruiker van het programma.
Topscore: Een score die gelijk of hoger is dan de laagste score in de Topscorelijst.

Topscorelijst: Een lijst met combinaties van scores en namen.

1.3 Overview

De rest van dit document is opgesplitst in vier delen.

In hoofdstuk 2 is een algemene omschrijving van het programma te vinden. Hierin worden onder andere het doel en de gebruikersomgeving beschreven. Tevens worden hier de logische modellen en sequentiediagrammen nader toegelicht.

In hoofdstuk 3 is een prototype voor de GUI te vinden. Dit prototype laat tevens ook eisen zien die niet gemakkelijk in het model te verwerken zijn.

In hoofdstuk 4 zijn de specifieke software-eisen beschreven. Dit is verder onderverdeeld in functionele en niet-functionele eisen. De functionele eisen komen voort uit het logische model. De niet-functionele eisen beschrijven eisen die niet gemakkelijk in het model te verwerken zijn.

Tenslotte staan in hoofdstuk 5 de traceability tabellen, welke een link vormen tussen SR's en hun bijbehorende UR(s) en andersom

2 Algemene omschrijving

2.1 Beschrijving van het model

2.1.1 Initieel model

In bijlage A kan het initiële model worden gevonden voor het *Etaris* spel. Het initiële model geeft een versimpelde weergave van *Etaris*, opgebouwd vanuit het URD. Klassen en methoden zijn afgeleid vanuit user requirements en weergegeven in UML. Er wordt nog geen rekening gehouden met de implementatie van programma en alles is zo eenvoudig mogelijk weergegeven.

Het model kan gezien worden als eigenlijk drie aparte packages.

Het gedeelte dat het werkelijke spel aanstuurt kan worden samengevat met de klassen *Game, Timer, PlayingField, Piece, Row, Block* en *Cell. Game* is de motor van het spel, en zal de connectie vormen tussen de andere twee gedeeltes van het spel. Verder zal *PlayingField* ieder van de twee speelvlakken van het spel aansturen. Deze speelvelden bestaan uit rijen (*Row*). Deze rijen kunnen gevuld zijn met blokken (*Block*) van een speelstuk (*Piece*) of gewoon leeg. Of ze opgevuld zijn of niet wordt bijgehouden door *Cell*. De beweging van het actieve speelstuk wordt aangeroepen via de *Game* class, met de functie *GetKeyPressed()* die daarna te bewegen richting doorstuurt naar het speelstuk (*Move()*).

Het tweede gedeelte van het spel zijn de menu's. Dit bestaat uit *MainMenu, OptionMenu* en *PauseMenu*. Het hoofdmenu is het scherm wat verschijnt als het programma wordt opgestart en is het menu wat gebruikt kan worden om een nieuw spel te starten. Het optiemenu wordt gebruikt om instellingen in te laden, zoals een nieuw configuratiefile (*GetConfig()*). Het pauzemenu dient om uit een actief spel te gaan, en om het spel natuurlijk te pauzeren.

Het laatste gedeelte is voor het bijhouden van de score. Hiervoor zijn *Score* en *ScoreFile* verantwoordelijk.

2.1.2 Sequentie diagrammen

De sequentiediagrammen geven een procesweergave van een aantal use cases.

De volgende cases zijn gemodelleerd:
Sequence chart 1: Het bewegen van een speelstuk (Bijlage B)

Sequence chart 2: Het laten vallen van een speelstuk met timer (Bijlage C)
Sequence chart 3: Het opstarten van het spel en het activeren van het optiemenu. (Bijlage D)
Sequence chart 4: Het ophogen van de score en game-over gaan. (Bijlage E)

Sequence chart 1
Het eerste sequentiediagram beschrijft het gedrag van het programma als in een actief spel de rechterpijltjestoets wordt ingedrukt. De speler drukt op de toets, waarna de keycode wordt doorgestuurd naar de klasse *Game* met de functie *GetKeyCode()*. Game roept vervolgens de functie *MovePiece(right)* van *PlayingField* aan. *PlayingField* roept de functie *Move(right)* van *Piece* aan. Vervolgens roept *Piece* voor de twee blokken van het speelstuk de functie *CanMove(right)* van *Block* aan. *Block* roept vervolgens de functie *GetIsOccupied()* aan van de gewenste *Cell*. Deze retourneert *false* voor beide cellen omdat de cellen rechts van de beide blokken vrij zijn. *Block* retourneert dezelfde waarde, *false*, naar *Piece*. Als *Piece* van elk *Block* de waarde *false* heeft ontvangen, roept *Piece* voor elk *Block* de functie *Move(right)* aan. Elk *Block* roept voor de gewenste *Cell* de functie *SetIsOccupied(true)* aan. Vervolgens retourneert *Piece true* naar het *PlayingField*. Deze roept een *Paint()* aan en retourneert *true* naar de klasse *Game*.

Sequence chart 2
Het tweede sequentiediagram beschrijft het gedrag als in het spel de timer naar de volgende stap gaat. *Timer* roept de functie *NextStep()* van *Game* aan. Vervolgens wordt door *Game* de functie *MovePiece(down)* van *PlayingField* aangeroepen. *PlayingField* roept *Move(down)* aan van *Piece*. *Piece* roept *CanMove(down)* aan van zijn eerste *Block*. *Block* roept *GetIsOccupied()* aan van *Cell*. *Cell* retourneert *false* omdat de cel onder het blok niet bezet is. *Block* retourneert *false* aan *Piece*. *Piece* roept *CanMove(down)* aan van zijn tweede en laatste *Block*. *Block* roept *GetIsOccupied()* van *Cell* aan. *Cell* retourneert *true* naar *Block* omdat de cel onder dit blok wel bezet is en *Block* retourneert *true* naar *Piece*. *Piece* heeft de waarde *true* teruggekregen dus kan het speelstuk niet naar beneden verplaatst worden. *Piece* retourneert *false* naar *PlayingField* om aan te geven dat het actieve speelstuk gestopt is met vallen. *PlayingField* roept *GetIsFilled()* aan van alle *Row*s. Deze retourneren *false* omdat deze rijen niet vol zijn. De stukken liggen stil en dus roept *PlayingField GetIsGameOver()* aan van *Piece*. Deze retourneert *false* omdat het speelstuk zich in zijn geheel in het speelveld bevindt. *Piece* roept *DeactivatePiece()* aan zodat straks een nieuwe *Piece* het spel kan binnenkomen. *PlayingField* retourneert *false* aan *Game* voor het niet naar beneden kunnen verplaatsen van het actieve speelstuk en field1 wordt inactief. *Timer* roept *NextStep()* aan. *Game* roept *SetActiveField(field2)* aan zodat het andere veld actief wordt.

Sequence chart 3
Het derde sequentiediagram beschrijft hoe het spel door de menu's loopt. Eerst wordt level 5 gekozen met *ChooseLevel(5)*, een functie van *MainMenu* aangeroepen door de gebruiken door het level te kiezen. Vervolgens klikt de speler op Start game, waarmee de functie *StartGame()* van *MainMenu* wordt aangeroepen. *MainMenu* roept de functie *SetActiveField(field1)* aan van *Game*. Game start vervolgens de timer met *StartTimer()*. Vervolgens wordt er een eerste speelstuk gekozen met de functie *GetPiece()* van *PlayingField*. Daarna roept *PlayingField* de functie *InsertPiece()* aan waarna *Paint()* wordt aangeroepen waardoor het eerste blokje het scherm in kan komen. De speler drukt op escape. *GetKeyPressed(esc)* van *Game* wordt aangeroepen. Hierdoor wordt de functie *PauseGame()* van *Game* aangeroepen. De timer wordt gestopt met *StopTimer()*. Het PauseMenu wordt geopend met *GoToPause()* van *PauseMenu*. De speler klikt op options waardoor het OptionsMenu wordt geopend door *GoToOptions()*. De speler klikt op annuleren en *GoToPrevious()* wordt aangeroepen. Vervolgens klikt de speler op end game en de functie *EndGame()* van *Game* wordt aangeroepen om het spel te beeindigen. Na het beeindigen wordt het MainMenu geopend met *GoToMain()*.

Sequence chart 4

In het vierde en laatste sequentiediagram wordt aangegeven hoe de score wordt bijgehouden en hoe het spel game over gaat. Dit diagram toont hoe het spel verloopt als een van de rijen wel gevuld wordt. Nadat de waarde *true* is geretourneerd van de functie *GetIsFilled()*, wordt aan de volle rij de score gevraagd met de functie *GetScore()*. Deze functie van *Row* retourneert de score van de rij. Vervolgens wordt de score verhoogd door de functie *IncreaseScore(score)* op te roepen. Hierna wordt de rij leeggemaakt met behulp van *Clear()*. Vervolgens roept *PlayingField Paint()* aan. De functie *Drop()* wordt aangeroepen om stukken eventueel te laten vallen na het verwijderen van de rij. Daarna wordt de functie *Drop()* nogmaals aangeroepen omdat er zich twee stukken boven de zojuist verwijderde rij bevinden. Nadat deze twee stukken indien mogelijk zijn gevallen, wordt weer gecheckt of er een *Row* vol is met *GetIsFilled()*. Dit is niet het geval en dus liggen de speelstukken stil en kan het volgende stuk worden ingevoegd. De functie *InsertPiece()* wordt aangeroepen. Nadat het actieve speelstuk niet verder kan vallen/stijgen, wordt gekeken of er een *Row* gevuld is. Dit is niet het geval. Daarna roept *PlayingField* de functie *GetIsGameOver()* aan van *Piece*. Deze functie retourneert *true* omdat het laatste speelstuk zich niet in zijn geheel in het speelveld bevindt en dus wordt de functie *SendScore(score)* van *Game* aangeroepen om de score op te slaan en wordt uiteindelijk de functie *EndGame()* van *Game* gebruikt om het spel te beeindigen.

2.1.3 Uitgebreid logisch model

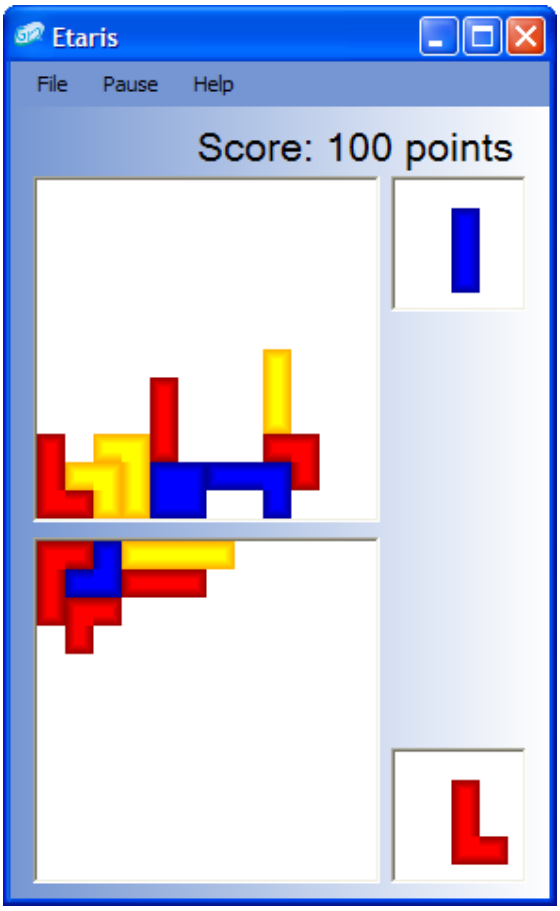
Als gevolg van deze sequence charts kan geconcludeerd worden dat maar een beperkt aantal wijzigingen nodig is om aan het uitgebreide model te komen. Veel van de functionaliteit was al voorzien in het URD, waardoor het initiële model maar een enkele aanpassing nodig heeft.

Het initiële model is uitgebreid met de klasse *DoneMove().* Deze klasse is toegevoegd zodat het spel weet wanneer een speelstuk klaar is met bewegen.

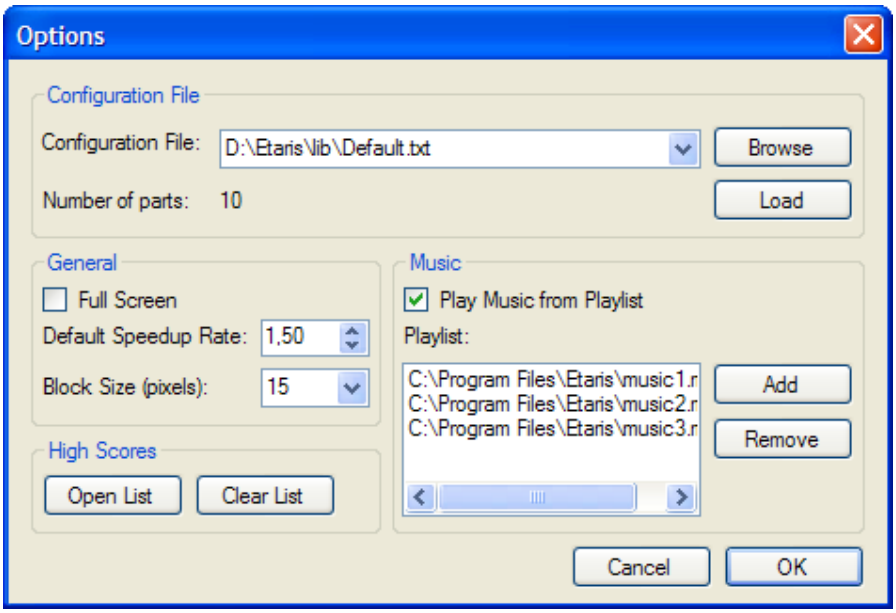
Verder is er nog een klasse toegevoegd voor het weergeven van het highscore menu; *HighScoreMenu*.

3 Grafische gebruikersinterface

3.1 Speelscherm

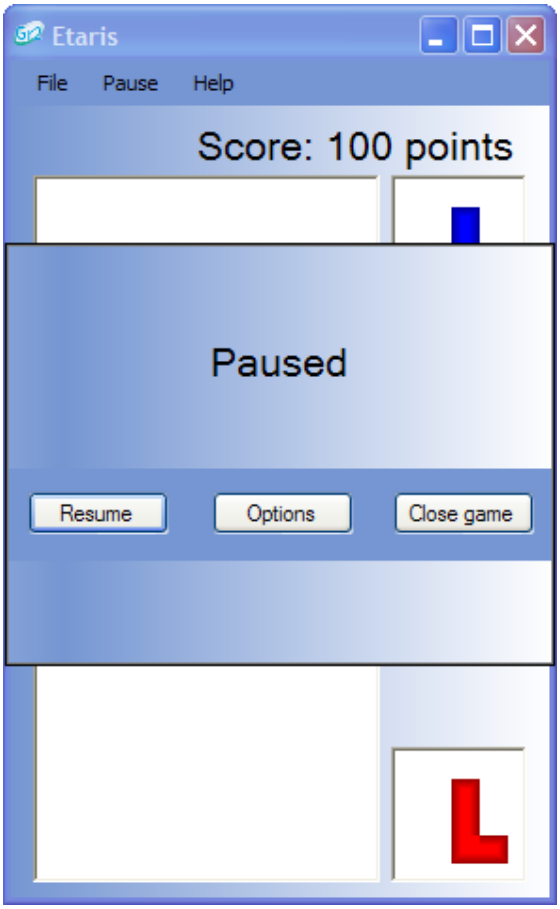


3.2 Optiemenu

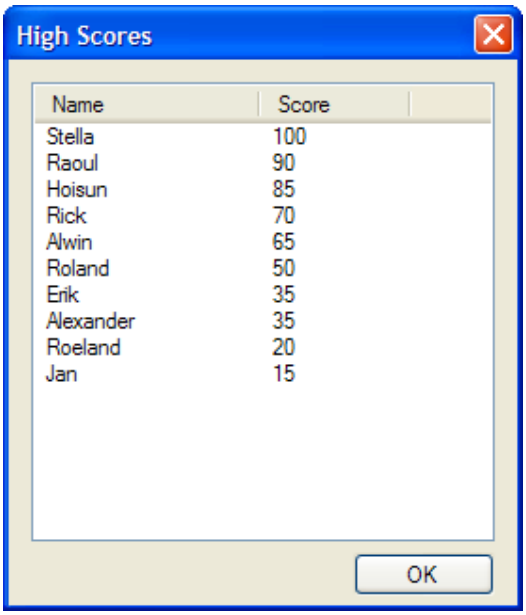


3.3 Hoofdmenu

3.4 Pauzemenu



3.5 Highscoremenu



4 Specifieke eisen

4.1 Functionele eisen

4.1.1 Block

SR-10 Block

Deze klasse houdt de kleur van de blokken bij.

Attributen

SR-11 blockColor: color

De kleur van het blok.

SR-p1 bCell: Cell

Een Block bezet cell bCell.

SR-81 blockX: int

De x coördinaat van een Block binnen zijn Piece.

SR-82 blockY: int

De y coördinaat van een Block binnen zijn Piece.

Methoden

SR-12a SetBlockColor(bcolor: color)

Kent het blok de kleur *color* toe.

Pre:

Post:
1. *blockColor* heeft de waarde van *bcolor*.

SR-12b CanMove(direction: string): boolean

Checkt de mogelijkheid voor verplaatsing van het blok, een cel in de richting *direction*.

Pre:

Post:
1. Als in die richting verplaatst kan worden is *true* geretourneerd, anders *false*.

SR-12c Move(direction: string): boolean

Verplaatst het blok een cel in de richting *direction*

Pre:
1. Het is mogelijk om het speelstuk in de gewenste richting te verplaatsen.

Post:
1. *Block* is verplaatst in de gewenste richting.

4.1.2 Cell

SR-13 Cell

Deze klasse houdt bij of de cellen bezet zijn.

1. *xPosition* is groter of gelijk aan 0.
2. *xPosition* is kleiner of gelijk aan 100.

Attributen

SR-14 occupied: boolean

Geeft aan of de cel bezet is.

SR-15 xPosition: int

Geeft aan op welke x-positie de cel zit.

SR-p7 cCell: cell
nCell: cell

Cell cCell kent zijn 4 buren nCell.

Methoden

SR-16 SetIsOccupied(occup: boolean): boolean

Verandert de waarde van *occupied* naar *occup*.

Pre:

Post:
1. *occupied* heeft de waarde van *occup* gekregen.

SR-17 SetPosition(x: int): void

Stelt *xPosition* in voor de cell.

Pre:
1. *x* is groter of gelijk aan 0 en kleiner of gelijk aan 100.

Post:
1. *xPosition* heeft de waarde van *x* gekregen.

4.1.3 Piece

SR-18 Piece

Bevat functies voor het bewegen en instellen van een speelstuk.

Attributen

SR-19 color: Color

De kleur van een speelstuk.

SR-p2 pBlock: Block

Een Piece bestaat uit meerdere pBlock: Blocks.

SR-83 pieceX: int

De x coördinaat van een Piece binnen zijn PlayingField.

SR-84 pieceY: int

De y coördinaat van een Piece binnen zijn PlayingField.

Methoden

SR-20a Move(direction: string): void

Verplaatst het speelstuk in de richting *direction*.

Pre:
1. Het speelstuk bevindt zich in een positie waarin het mogelijk is om een blokje in de richting *direction* te bewegen.

Post: 1. Het speelstukis een blok in de richting *direction* verplaatst.

SR-20b DoneMove(): void

Geeft aan dat het actieve speelstuk klaar is met bewegen.

Pre: 1. Het speelveld is net opnieuw getekend en uitgetekend.

Post: 1. Het speelstuk is in de nieuwe positie en de speler kan een nieuwe beweegrichting aangeven.

SR-21 Turn(): void

Draait het speelstuk een kwartslag met de klok mee.

Pre: 1. Het speelstuk bevindt zich op een positie waar hij vrij kan draaien.

Post: 1. Het speelstuk is een kwartslag met de klok mee gedraaid.

SR-22 Drop(): void

Laat het speelstuk naar beneden vallen.

Pre: 1. Het speelstuk steunt nergens op.

Post: 1. In het bovenste speelveld is het speelstuk zo ver mogelijk omlaag gevallen en leunt nu op een ander speelstuk of op de bodem van het speelveld.
2. In het onderste speelveld is het speelstuk zo ver mogelijk omhoog gevallen en leunt nu op een ander speelstuk of op de bovenkant van het speelveld.

SR-23 DeactivatePiece(): void

Deactiveert het speelstuk.

Pre: 1. IsActive = True

Post: 1. IsActive = False

SR-24 GetGameOver(): Boolean

Kijkt of het spel game-over is.

Pre: 1. Het actieve speelstuk kan niet meer verder vallen en heeft al zijn bewegingen gemaakt voor deze kloktik.

Post: 1. Als het speelstuk niet meer binnen de randen van het speelveld past, dan geeft *GetGameOver true* terug.
2. Als het speelstuk wel binnen de randen van het speelveld past, dan geeft *GetGameOver false* terug.

4.1.4 Row

SR-25 Row

Deze klasse houdt bij welke cellen bij een rij horen en bevat functies voor het wissen van rijen.

1. *width* is altijd groter of gelijk aan 0.

Attributen

SR-26 width: int

Geeft de breedte van het speelveld aan in blokken.

SR-27 isFull: boolean

Geeft aan of de rij volledig gevuld is.

SR-28 yPosition: int

Geeft de y-positie van de rij aan.

SR-p4 rCell: Cell

Een Row bestaat uit 1 tot 100 Cellen rCell.

Methoden

SR-29 Clear(): void

Maakt de rij leeg.

Pre:	1. *GetIsFilled* geeft de waarde *true*.
Post:	1. *Row* is volledig leeggemaakt.

SR-30 GetIsFilled(): boolean

Kijkt of de rij volledig gevuld is.

Pre:	1. Het actieve speelstuk kan niet meer bewegen.
Post:	1. Als de rij vol is, dan geeft *CheckIsFilled true* terug. 2. Als de rij niet vol is, dan geeft *CheckIsFilled false* terug.

SR-31 SetScore(): void

Geeft punten voor het verwijderen van de rij.

Pre:	1. *GetIsFilled* geeft de waarde *true*. 2. Volle rijen zijn nog niet verwijderd deze kloktik.
Post:	1. De toename van de score is verhoogd als gevolg van het aantal te verwijderen rijen.

4.1.5 Score

SR-32 Score

Deze klasse houdt de score bij die wordt gehaald.

1. De score is altijd groter of gelijk aan 0.

Attributen

SR-33 scoreHeight: int

Geeft aan hoe hoog de score is.

SR-p5 sScore: ScoreFile

Scores worden opgeslagen in ScoreFile sScore.

Methoden

SR-34 ClearScore(): void

Verwijdert de score.

Pre:	1. De speler heeft een score behaald.
Post:	1. De behaalde score is verwijderd.

SR-35 IncreaseScore(tempScore: int): void

Verhoogt de score met *tempScore*.

Pre:
1. *tempScore* is positief.

Post:
1. De score is verhoogd met *tempScore*.

SR-36 SaveScore(): void

Slaat de score op.

Pre:
1. De speler heeft een score behaald.

Post:
1. De behaalde score is opgeslagen in de scorelijst.

4.1.6 PlayingField

SR-38 PlayingField

Deze klasse houdt bij welk speelveld actief is en welk speelstuk als volgende verschijnt.

Attributen

SR-39 fieldID: int

Geeft aan welk speelveld het is.

SR-40 height: int

Geeft de hoogte van het speelveld aan.

SR-41 active: boolean

Geeft aan of het speelveld actief is.

SR-p9 pfPiece: Piece
pField: Field

Een PlayingField pField kent meerdere Piece pfPiece

SR-p10 pRow: Row

Een PlayingField bestaat uit 1 tot 100 rijen pRow.

Methoden

SR-42a GetPiece(): void

Kiest een willekeurig speelstuk uit de beschikbare speelstukken en laat dit zien in het preview veld van het speelscherm.

Pre:
1. *InsertPiece()* is deze kloktik al aangeroepen.

Post:
1. Er is een nieuw speelstuk gekozen.
2. Dit speelstuk wordt een nieuwe kleur toegekend, gekozen uit de mogelijk te kiezen kleuren.
3. Dit speelstuk wordt getoond in het preview veld van het speelscherm.

SR-42b InsertPiece(): void

Stopt het speelstuk gekozen door *GetPiece* in het speelveld.

Pre:	1. Er is een speelstuk gekozen door *GetPiece.*
Post:	1. Het nieuwe speelstuk komt in het midden van het actieve speelveld binnen.

SR-43 MovePiece(direction: string): void

Geeft een speelstuk de opdracht om in een bepaalde richting te bewegen, indien mogelijk.

Pre:	1. Er is een actief speelstuk dat nog niet rust op een ander speelstuk.
Post:	1. *Move* is aangeroepen, met de richting waarin het speelstuk zich moet bewegen.

SR-44 TurnPiece(): void

Vertelt een actief speelstuk om te draaien, indien mogelijk.

Pre:	1. Er is een actief speelstuk dat nog niet rust op een ander speelstuk.
Post:	1. *Turn()* is aangeroepen in het actieve speelstuk.

4.1.7 Game

SR-45 Game

De 'motor' van het spel, deze klasse houdt de besturing van het actieve spel bij en regelt de verhoging van de score van de speler.

1. Als de game klasse actief is, dan draait het spel
2. Er is altijd een speelveld actief.

Attributen

SR-46 tempScore: int

Bewaart de score verzamelt tijdens 1 tik van de Timer klasse.

SR-p13 gField: PlayingField
pfGame: Game

Een spel pfGame bestaat uit 2 PlayingField: gField.

Methoden

SR-47 GetKeyPressed(kc: keycode): void

Detecteert of een toets is ingedrukt door de speler, en zal als gevolg hiervan de juiste actie uitvoeren.

Pre:	1. Er is een actief speelstuk.
Post:	1. Als de keycode bij een pijltjestoets hoort voert het actieve *Piece MovePiece()* uit met de gegeven *direction*. 2. Als de keycode bij de enter toets hoort voert het actieve *Piece Turn()* uit. 3. Als de keycode bij de escape toets hoort wordt de functie *PausedGame()* aangeroepen.

SR-48 GetOptions(): void

GetOptions haalt de opties op die zijn ingesteld via het optiemenu.

Pre:	1. Er zijn nog geen opties ingeladen.
Post:	1. De opties ingesteld in het optiemenu zijn ingeladen.

SR-49 SendScore(tScore: int)

Stuurt de huidige waarde van *tempScore* door.

Pre:	1. De waarde van *tempScore* is deze tik nog niet doorgestuurd naar de totale spelersscore.
Post:	1. De totale spelersscore is verhoogd met de waarde van *tempScore*. 2. De waarde van *tempScore* is gelijk aan 0.

SR-50 GetPieces(): void

Laadt de speelstukken in die zijn aangegeven in de configuratiefile.

Pre:	1. Er zijn nog geen speelstukken ingeladen.
Post:	1. De speelstukken aangegeven in de configuratiefile zijn ingeladen.

SR-51 SetActivateField(field: int)

Bepaalt welk speelveld actief is.

Pre:	1. Er is 0 of 1 speelveld actief.
Post:	1. Als er 0 speelvelden actief zijn voor het aanroepen van *SetActivateField*, dan is speelveld 1 actief na het aanroepen. 2. Als het vorige actieve speelveld 1 was, is het actieve speelveld nu 2. 3. Als het vorige actieve speelveld 2 was, is het actieve speelveld nu 1.

SR-52 PauseGame(): void

Pauzeert het spel

Pre:	1. Het spel is niet gepauzeerd.
Post:	1. Het spel is gepauzeerd. 2. Het spel laat het pauzemenu zien.

SR-85 StartGame(): void

Start een nieuw spel.

Pre:	1. Er is nog geen Game actief.
Post:	1. Het spel is gestart.

SR-53a EndGame(): void

Beëindigt het spel als het game-over is.

Pre:	1. De functie *GetGameOver* geeft een *true*.
Post:	1. Het spel is beëindigd. 2. Het speelveld laat “GAME OVER!” zien. 3. Speler krijgt de mogelijkheid zijn topscore in te voeren als deze tot de beste 10 behoort. 4. De speler krijgt de topscorelijst te zien.

SR-53b Repaint(): void

Tekent het speelscherm opnieuw.

Pre:	
Post:	1. Het speelscherm is opnieuw getekend, in de toestand ontstaan na het verwerken van alle informatie die de deze kloktik is gegenereerd.

4.1.8 Timer

SR-54 Timer

De *Timer* klasse houdt de ticker bij die er voor zorgt dat het Etaris spel verder gaat naar de volgende stap in de tijd.

1. De ticker maakt altijd een stap op de plaats of 1 stap vooruit.
2. De tijdswaarde van de ticker is een positief en geheel getal.
3. De speed-up-rate heeft altijd een positieve en reële waarde.

Attributen

SR-55a speed: int

De speed attribuut geeft aan hoe snel de ticker tikt in de Timer klasse.

SR-p14 tGame: Game
gTimer: Timer

Timer gTimer beïnvloed Game tGame.

Methoden

SR-55b StartTimer(): void

Start de timer.

Pre:

Post:
1. De timer loopt vanaf het vorig opgeslagen tijdstip, indien er geen vorig tijdstip beschikbaar is, is vanaf tijdstip 0.

SR-55c StopTimer(): void

Pauzeert de timer.

Pre:

Post:
1. De timer loopt niet.
2. De huidige waarde van de *Timer* is opgeslagen zodat vanaf dat punt de kloktikken hervat kunnen worden.

SR-55d NextStep(): void

Gaat naar de volgende tijdsstap.

Pre:
1. De juiste hoeveelheid tijd is verstreken tussen de deze en de vorige stap.

Post:
1. Het spel is doorgegaan naar de volgende tijdsstap.

SR-55e SpeedUpRate(currentSpeed: int; upSpeed: int): int

Bepaalt met welke snelheid de tiksnelheid toeneemt.

Pre:
1. speed = oldSpeed

Post:
1. speed = oldSpeed + upSpeed.

4.1.9 OptionMenu

SR-56 OptionsMenu

De OptionsMenu klasse houdt alle functies bij die in het optiemenu van *Etaris* plaatsvinden.

Attributen

SR-57 mustPlayMusic: boolean

mustPlayMusic geeft aan of er wel (in het geval van true), of geen (in het geval van false) gespeeld moet worden tijdens het spel.

SR-p12 mScore: Score

De score mScore kan gewist worden vanuit het optiesmenu.

SR-p18 mOption: OptionMenu
mMain: MainMenu

Het MainMenu mMain kan het OptionMenu mOption aanroepen en het OptionMenu mOption kan het MainMenu mMain aanroepen.

SR-p19 mOption: OptionMenu
mPause: PauseMenu

Het PauseMenu mPause kan het OptionMenu mOption aanroepen en het OptionMenu mOption kan het PauseMenu mPause aanroepen.

SR-p20 cFig: ConfigFile

Het OptionMenu laadt de configfile cFig.

SR-p21 cMusic: Music

Het optiemenu laadt Music cMusic.

Methoden

SR-58 GetConfig(): void

Deze methode laadt de configuratie file in en controleert of deze geldig is.

Pre:
1. Het spel gebruikt de configuratie die is aangegeven via *GetConfig*, als er geen configuratie is aangegeven, dan gebruikt het spel een standaard configuratie.

Post:
1. De nieuwe configuratiefile wordt gebruikt door het spel voor de veldgrootte en de beschikbare speelblokken mits de grootte van het speelveld in de nieuwe configuratie file is aangegeven in positieve en gehele getallen, groter of gelijk aan 1.
2. De nieuwe configuratiefile wordt gebruikt door het spel voor de veldgrootte en de beschikbare speelblokken mits de breedte van het speelveld kleiner of gelijk is aan 100.
3. De nieuwe configuratiefile wordt gebruikt door het spel voor de veldgrootte en de beschikbare speelblokken mits de hoogte van het speelveld kleiner of gelijk is aan 100.
4. De nieuwe configuratiefile wordt gebruikt door het spel voor de veldgrootte en de beschikbare speelblokken mits alle speelstukken geheel binnen een leeg speelveld passen.
5. Als 4. overtreden wordt dan krijgt de gebruiker de keuze om te kiezen alsnog de configuratiefile in te laden. Als hij “Yes” kiest wordt de nieuwe file ingeladen, als hij “No” kiest, wordt de oude file ingeladen.
6. Als 1 t/m 3 worden overtreden, dan blijft het vorige configuratiefile ingeladen.
7. De na het inladen aangegeven speelstukken worden getoond aan de speler.

SR-59 PlayMusic(on: boolean): void

Deze methode bepaalt of er wel of geen muziek gespeeld wordt door de waarde van de attribuut *mustPlayMusic* aan te passen via de boolean "*on*".

Pre:	1. Het attribuut *mustPlayMusic* staat op true of false
Post:	1. Als de waarde van *on* op *false*, dan staat na de methode *mustPlayMusic* op *false* en de muziek speelt niet meer. 2. Als de waarde van *on* op *true*, dan staat na de methode *mustPlayMusic* op *true* en de muziek speelt.

SR-60 GoToPrevious(): void

Deze methode stuurt de speler terug naar het speelveld waar hij vandaan kwam voordat hij naar het optiesmenu kwam.

Pre:	1. Het spel laat het optiesmenu zien.
Post:	1. Het optiesmenu is afgesloten. 2. Het spel laat het scherm zien waar de speler zich in bevond voordat hij naar het optiesmenu ging.

SR-61a SetBlockSize(size: int): void

Deze functie verandert de standaardgrootte van de blokken gebruikt in het spel in pixels. De nieuwe lengte en breedte worden gelijk aan *size*.

Pre:	1. De blokken hebben minstens een grootte van 15 bij 15 pixels
Post:	1. De blokken hebben minstens een grootte van 15 bij 15 pixels 2. Als 1 en 2 worden overtreden dan hebben de blokken de vorige geldige grootte.

SR-61b ClearAllScores(): void

Wist alle scores.

Pre:	1. Er zijn 10 scores ingeladen.
Post:	1. Alle scores zijn vervangen met dummyscores.

4.1.10 MainMenu

SR-62 MainMenu

De *MainMenu* klasse houdt alle functies bij die in het hoofdmenu van *Etaris* plaatsvinden.

Attributen

SR-p11 mScore: ScoreFile

Het MainMenu kan de ScoreFile mScore laten zien.

SR-p15 mGame: Game

Het hoofdmenu start het Game mGame.

SR-p16 mTimer: Timer

Hoofdmenu bepaald met welke snelheid de Timer mTimer het spel begint.

SR-p18 mOption: OptionMenu
mMain: MainMenu

Het MainMenu mMain kan het OptionMenu mOption aanroepen en het OptionMenu mOption kan

het MainMenu mMain aanroepen.

SR-p22 mMain: MainMenu
mHigh: HighScoreMenu

Het MainMenu mMain kan het HighScoreMenu mHigh aanroepen en het HighScoreMenu mHigh kan het MainMenu mMain aanroepen.

Methoden

SR-63a ChooseLevel(levelspeed: int): void

Deze methode bepaald op welke snelheid de ticker begint met tikken.

Pre:	1. De *speed* variabele uit de *Timer* klasse staat op 0.
Post:	1. De *speed* variabele uit de *Timer* klasse heeft de waarde aangenomen van de *levelspeed* variabele uit de *ChooseLevel* methode.

SR-63b SetSpeedUp(upSpeed: int): int

Stelt de speed-up rate in door deze door te sturen naar de *Game* class.

Pre:	1. De speed-up rate heeft een waarde.
Post:	1. De speed-up rate heeft de waarde van *upSpeed* aangenomen.

SR-64 StartGame(): void

Start het spel.

Pre:	1. De *speed* variabele uit de *Timer* klasse heeft een waarde niet gelijk aan 0. 2. Er is een geldig configuratiefile ingeladen. 3. Alle opties in het optiemenu zijn ingesteld.
Post:	1. De speler bevindt zich nu in het speelscherm. 2. De instellingen uit het optiemenu worden gebruikt tijdens het spel. 3. De ticker uit de *Timer* klasse is gestart.

SR-65a GoToOptions(): void

Stuurt de speler naar het opties menu.

Pre:	1. De speler bevindt zich niet in het optiemenu
Post:	1. De speler bevindt zich in het optiemenu.

SR-65b ShowScores(): void

Laat de speler de topscores zien.

Pre:	1. De speler bevindt zich in het hoofdmenu.
Post:	1. De speler ziet de topscores.

SR-66 ExitGame(): void

Sluit het programma af vanuit het hoofdmenu.

Pre:	1. Het programma *Etaris* draait. 2. De speler bevindt zich in het hoofdmenu.
Post:	1. De speler is gevraagd of hij het spel wil afsluiten. 2. Indien de speler “Yes” heeft gekozen, is het spel *Etaris* afgeloten. 3. Indien de speler “No” heeft gekozen, bevindt hij zich weer in het hoofdmenu.

4.1.11 PauseMenu

SR-67 PauseMenu

De PauseMenu klasse houdt alle functies bij die in het pauzemenu van *Etaris* plaatsvinden.

Attributen

SR-p17 mTimer: Timer

Het pauzemenu kan de Timer mTimer hervatten.

SR-p19 mOption: OptionMenu
mPause: PauseMenu

Het PauseMenu mPause kan het OptionMenu mOption aanroepen en het OptionMenu mOption kan het PauseMenu mPause aanroepen.

Methoden

SR-68 ResumeGame(): void

Hervat het spel nadat het gepauzeerd is.

Pre:	1. De speler bevindt zich in het pauzemenu.
Post:	1. De speler bevindt zich in het speelscherm. 2. Het speelscherm is in dezelfde toestand als voordat de speler naar het pauzescherm toe ging.

SR-69 EndPausedGame(): void

Eindigt het huidige spel, en brengt de speler terug naar het hoofdmenu.

Pre:	1. De speler bevindt zich in het pauzemenu.
Post:	1. De speler is gevraagd of hij het spel wil afsluiten. 2. Indien de speler “Yes” heeft gekozen, dan bevindt hij zich in het scorescherm, waarna hij verder kan gaan naar het hoofdmenu. 3. Indien de speler “No” heeft gekozen, dan bevindt hij zich in weer in het pauzemenu.

4.1.12 HighScoreMenu

SR-70a HighScoreMenu

De *HighScoreMenu* klasse houdt alle functies bij die in het HighScoreMenu van *Etaris* plaatsvinden.

Attributen

SR-p22 mMain: MainMenu
mHigh: HighScoreMenu

Het MainMenu mMain kan het HighScoreMenu mHigh aanroepen en het MainMenu mMain kan het HighScoreMenu mHigh aanroepen.

Methoden

SR-70b GoToHighScore(): void

Stuurt de speler naar het HighScoreMenu.

Pre:	1. De speler bevindt zich niet in het HighScoreMenu.
Post:	1. De speler bevindt zich in het HighScoreMenu.

4.1.13 ScoreList

SR-71 ScoreList

Houdt de scores bij.

Methoden

SR-72 LoadScore(): void

Laad alle opgeslagen scores.

Pre:

Post:
1. De laatst opgeslagen scorelijst is ingeladen, deze lijst heeft 10 scores.
2. Indien de lijst minder dan 10 scores bevat, is deze aangevuld met dummy scores.

SR-73 DeleteScore(delScore: int): void

Verwijdert de *n*de score. Met de *n* gelijk aan de waarde van *delScore*.

Pre:
1. Er zijn 10 scores ingeladen in de scorelijst.

Post:
1. De *n*de score is verwijderd.
2. Indien er minder dan 10 scores waren is een dummyscore ingevoegd onder aan de lijst zodat er weer 10 scores zijn.

SR-74 InsertScore(newScore: int, scoreName: string): void

Voegt een nieuwe score in.

Pre:
1. De nieuwe score is hoger dan de laagste van de 10 scores opgeslagen in de score lijst.

Post:
1. De nieuwe score is ingevoerd.
2. De speler zijn naam is ingevuld in de scorelijst.
3. De scorelijst is gesorteerd.
4. De laagste score in de nieuwe gesorteerde lijst is verwijderd, waardoor er weer 10 scores zijn in de scorelijst.

SR-75 SaveScoreList(): void

Slaat de huidige scorelijst op.

Pre:
1. Er is een scorelijst ingeladen.

Post:
1. De nieuwe scorelijst is opgeslagen in een .xml file.

4.2 Niet-functionele eisen

SR-76 *Etaris* is geschreven in C#.
SR-77 De gebruikershandleiding is geschreven in het Nederlands.
SR-78 Het commentaar in de code is geschreven in het Nederlands.
SR-79 De gebruikersinterface is in het Brits Engels.
SR-80 *Etaris* moet draaien op Windows XP met ".NET framework 3.x".

5 Traceability tabellen

5.1 UR naar SR

URD-requirement	Bijbehorend(e) SR-nummer(s)
UR-10	SR-62
UR-11	SR-58
UR-12	SR-58
UR-13	SR-24
UR-14	SR-24, SR-53
UR-15	SR-53
UR-16	SR-54, SR-20a, SR-20b
UR-17	SR-54, SR-20a, SR-20b
UR-19	SR-38, SR-39, SR-42a, SR-42b, SR-50
UR-20	SR-45, SR-51
UR-21	SR-51
UR-22	SR-27, SR-29, SR-30
UR-23	SR-27, SR-29, SR-30
UR-24	SR-22
UR-25	SR-22
UR-26	SR-22
UR-27	SR-22
UR-28	SR-31, SR-32, SR-33, SR-34, SR-36, SR-41
UR-29	SR-38, SR-42a
UR-30	SR-38, SR-42a
UR-31	SR-57, SR-59
UR-32	SR-56, SR-57, SR-59
UR-33	SR-56, SR-57, SR-59
UR-34	SR-71, SR-72, SR-73, SR-74
UR-35	SR-71, SR-72
UR-36	SR-71, SR-73
UR-37	SR-71, SR-74
UR-38	SR-71, SR-74
UR-39	SR-71, SR-74
UR-40	SR-71, SR-74
UR-41	SR-71, SR-74
UR-42	SR-71, SR-24
UR-43	SR-19, SR-42a
UR-44	SR-10, SR-11, SR-12a, SR-42b
UR-45	SR-42b
UR-46	SR-55e, SR-63b
UR-47	SR-63a
UR-48	SR-58
UR-49	SR-58
UR-50	SR-50, SR-58
UR-51	SR-50, SR-58
UR-52	SR-25, SR-26, SR-58
UR-53	SR-40, SR-58
UR-54	SR-50, SR-58
UR-55	SR-50, SR-58

URD-requirement	Bijbehorend(e) SR-nummer(s)
UR-56	SR-58
UR-57	SR-18, SR-44
UR-58	SR-18, SR-44
UR-59	SR-75
UR-60	SR-63a
UR-61	SR-50, SR-56, SR-58
UR-62	SR12b, SR-12c, SR-18, SR-20a, SR-20b, SR-43
UR-63	SR12b, SR-12c, SR-18, SR-20a, SR-20b, SR-43
UR-64	SR-21, SR-44, SR-67
UR-65	SR-52, SR-55c, SR-67
UR-66	SR-60, SR-67, SR-68
UR-67	SR-60, SR-67, SR-68
UR-68a	SR-56, SR-61a
UR-68b	SR-56, SR-61a
UR-68c	SR-56, SR-61a
UR-69	SR-65b
UR-70	SR-56
UR-71	SR-56
UR-72	SR-48, SR-56
UR-73	SR-65a
UR-74	SR-66
UR-75	SR-64
UR-77	SR-67, SR-70
UR-78	SR-67, SR-69
UR-79	SR-69
UR-80	SR-69
UR-81	SR-70
UR-82	SR-68
UR-85	SR-40
UR-86	SR-40
UR-87	SR-56
UR-90	SR-76
UR-91	SR-77
UR-92	SR-78
UR-93	SR-79
UR-94	SR-80
UR-95	SR-61a
UR-96	SR-61a

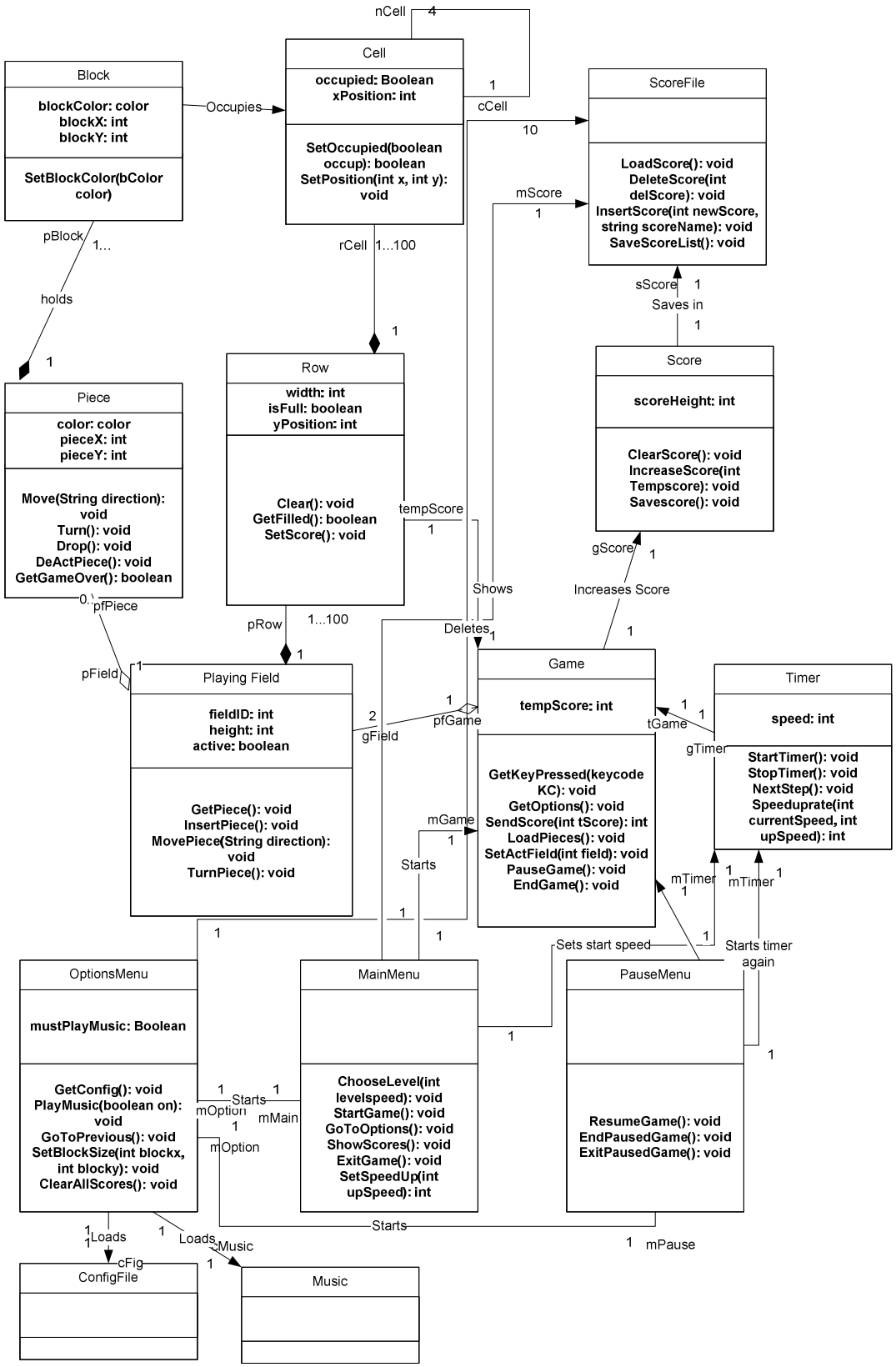
5.2 SR naar UR

SRD-requirement	Bijbehorend(e) UR-nummer(s)
SR-10	UR-44
SR-11	UR-44
SR-12a	UR-44
SR-12b	UR-62, UR-63
SR-12c	UR-62, UR-63
SR-13	UR-24, UR-25, UR-26, UR-27, UR-57, UR-62, UR-63
SR-14	UR-57

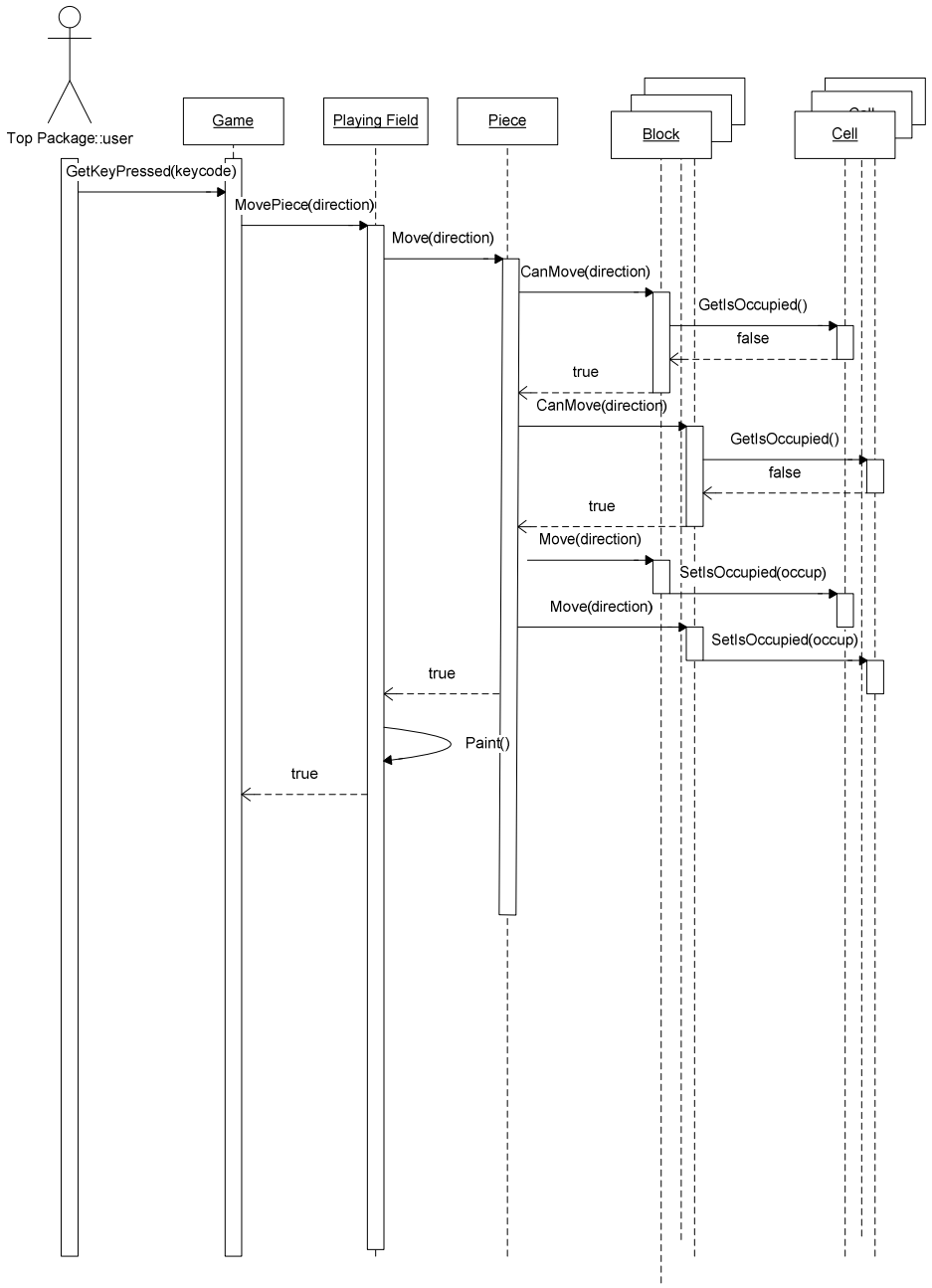
SRD-requirement	Bijbehorend(e) UR-nummer(s)
SR-15	UR-62, UR-63
SR-16	UR-24, UR-25, UR-26, UR-27, UR-62, UR-63
SR-17	UR-24, UR-25, UR-26, UR-27, UR-57, UR-62, UR-63
SR-18	UR-57, UR-58, UR-62, UR-63
SR-19	UR-43
SR-20a	UR-16, UR-17, UR-62, UR-63
SR-20b	UR-16, UR-17, UR-62, UR-63
SR-21	UR-64
SR-22	UR-24, UR-25, UR-26, UR-27
SR-23	UR-19, UR-26, UR-27
SR-24	UR-13, UR-14, UR-42
SR-25	UR-52
SR-26	UR-52
SR-27	UR-22, UR-23
SR-28	UR-24, UR-25
SR-29	UR-22, UR-23
SR-30	UR-22, UR-23
SR-31	UR-28
SR-32	UR-28
SR-33	UR-28
SR-34	UR-28
SR-35	UR-28
SR-36	UR-28
SR-38	UR-19, UR-20, UR-29, UR-30
SR-39	UR-19
SR-40	UR-83, UR-84, UR-85, UR-86, UR-53
SR-41	UR-28
SR-42a	UR-19, UR-29, UR-30, UR-43
SR-42b	UR-19, UR-44, UR-45
SR-43	UR-62, UR-63
SR-44	UR-57, UR-58, UR-64
SR-45	UR-20
SR-46	UR-28
SR-47	UR-62
SR-48	UR-72
SR-49	UR-28
SR-50	UR-19, UR-50, UR-51,UR-54, UR-55, UR-56, UR-61
SR-51	UR-20, UR-21
SR-52	UR-65
SR-53a	UR-14, UR-15
SR-53b	UR-22, UR-23, UR-24, UR-25, UR-26, UR-27, UR-29, UR-30, UR-45, UR-62, UR-63
SR-54	UR-16, UR-17
SR-55a	UR-46, UR-47
SR-55b	UR-46, UR-47
SR-55c	UR-46, UR-47UR-65
SR-55d	UR-46, UR-47
SR-55e	UR-46
SR-56	UR-32, UR-33, UR-61, UR-68a, UR-68b, UR-68c, UR-70, UR-71, UR-72, UR-87
SR-57	UR-31, UR-32, UR-33

SRD-requirement	Bijbehorend(e) UR-nummer(s)
SR-58	UR-11, UR-12, UR-48, UR-49, UR-50, UR-51, UR-52, UR-53, UR-54, UR-55, UR-56, UR-61
SR-59	UR-31, UR-32, UR-33
SR-60	UR-66, UR-67
SR-61a	UR-68a, UR-68b, UR-68c, UR-95, UR-96
SR-61b	UR-34, UR-35
SR-62	UR-10
SR-63a	UR-47, UR-60
SR-63b	UR-46
SR-64	UR-75
SR-65a	UR-73
SR-65b	UR-69
SR-66	UR-74
SR-67	UR-64, UR-65, UR-66, UR-67, UR-77, UR-78
SR-68	UR-66, UR-67
SR-69	UR-78
SR-70a	UR-34, UR-35, UR-36, UR-37, UR-38, UR-39, UR-40, UR-41, UR-42
SR-70b	UR-41, UR-42
SR-71	UR-34, UR-35, UR-36, UR-37, UR-38, UR-39, UR-40, UR-41, UR-42
SR-72	UR-34, UR-35
SR-73	UR-34, UR-36
SR-74	UR-34, UR-37, UR-38, UR-39, UR-40, UR-41
SR-75	UR-59
SR-76	UR-90
SR-77	UR-91
SR-78	UR-92
SR-79	UR-93
SR-80	UR-94

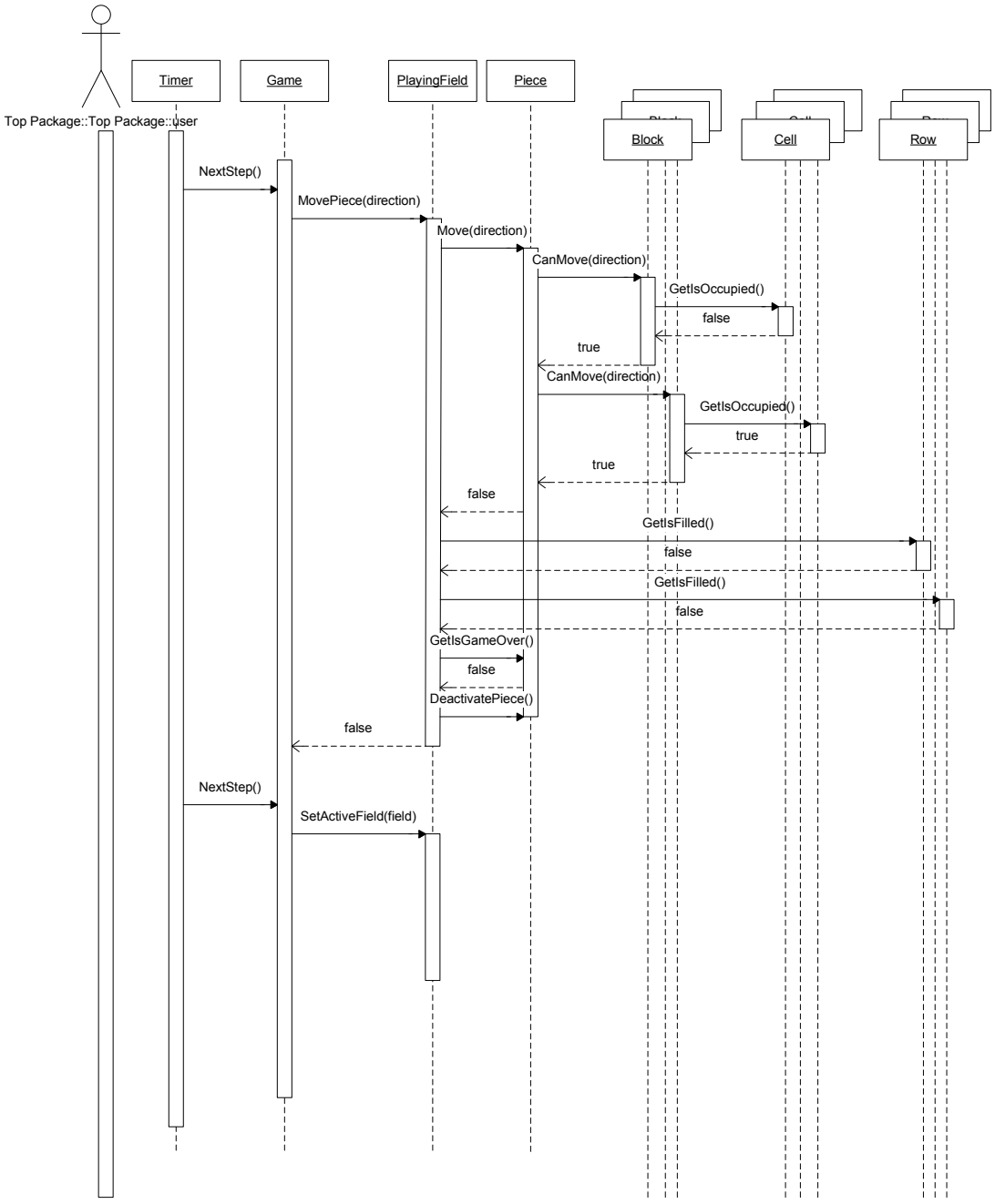
Bijlage A: Initiële model



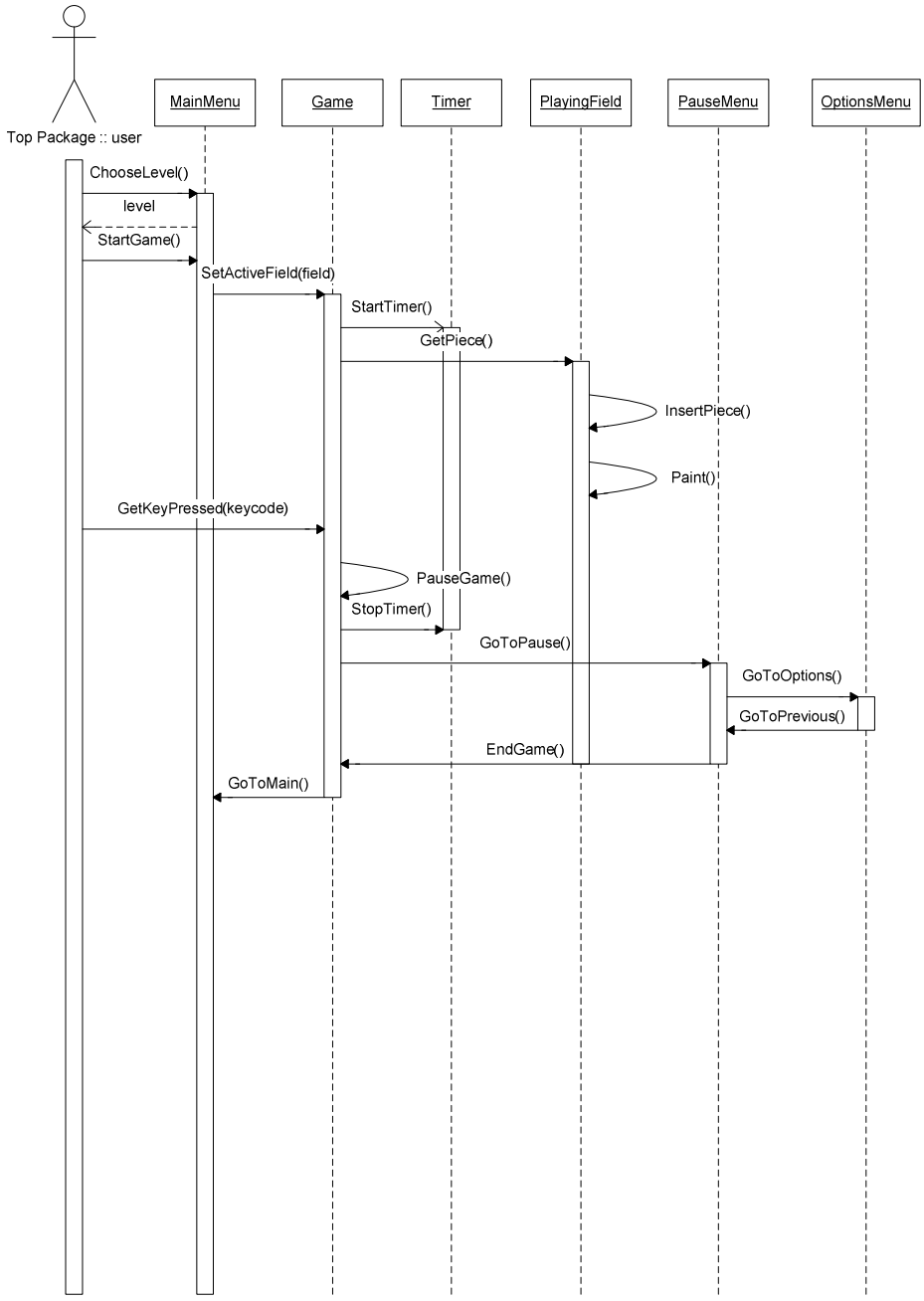
Bijlage B: Sequence chart 1: Het bewegen van een speelstuk



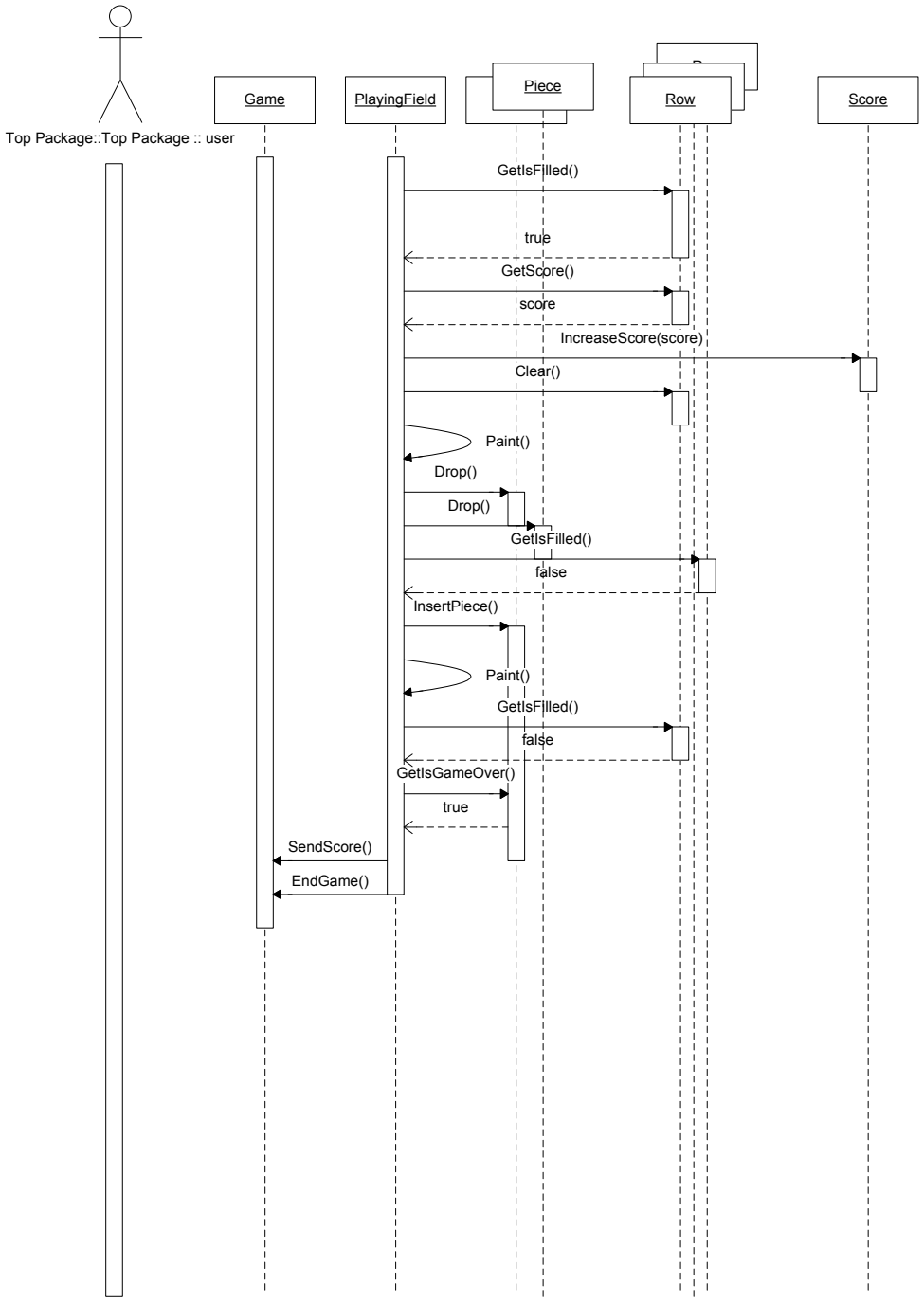
Bijlage C: Sequence chart 2: Het laten vallen van een speelstuk met timer



Bijlage D: Sequence chart 3: Het opstarten van het spel en het activeren van het optiesmenu.



Bijlage E: Sequence chart 4: Het ophogen van de score en game-over gaan.



Bijlage F: Extended Logic Model