FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

MULTIDISCIPLINAIR INGENIEURSPROJECT

TEAM 12: BATTLESHIP

Dylan De Roe
Robbe De Sutter
Karel Everaert
Dries Ryckbosch
Marie-Sophie Verwee

Academiejaar: 2017 - 2018

UNIVERSITEIT
GENT

Inhoudstafel

Inleiding

Tijdens dit multidisciplinair ingenieursproject moesten we in een team van vijf personen een niet-triviaal algoritme implementeren. De opdracht was om dit in Python te schrijven. Als project hebben we gekozen om zeeslag te maken (of in het Engels: Battleship). In onze implementatie zal de speler het telkens opnemen tegen een computer. Dit project zal een pariteitsalgoritme gebruiken om het spel na te bootsen (zie gedeelte Algoritme).

Wanneer het spel wordt opgestart, krijgt de speler een aantal keuzes. Hij krijgt de keuze om een nieuw spel te starten, een oud spel in te laden, om de regels te bekijken of om het spel af te sluiten. In afbeelding 1 wordt het startscherm weergegeven.

Afb. 1: Startscherm

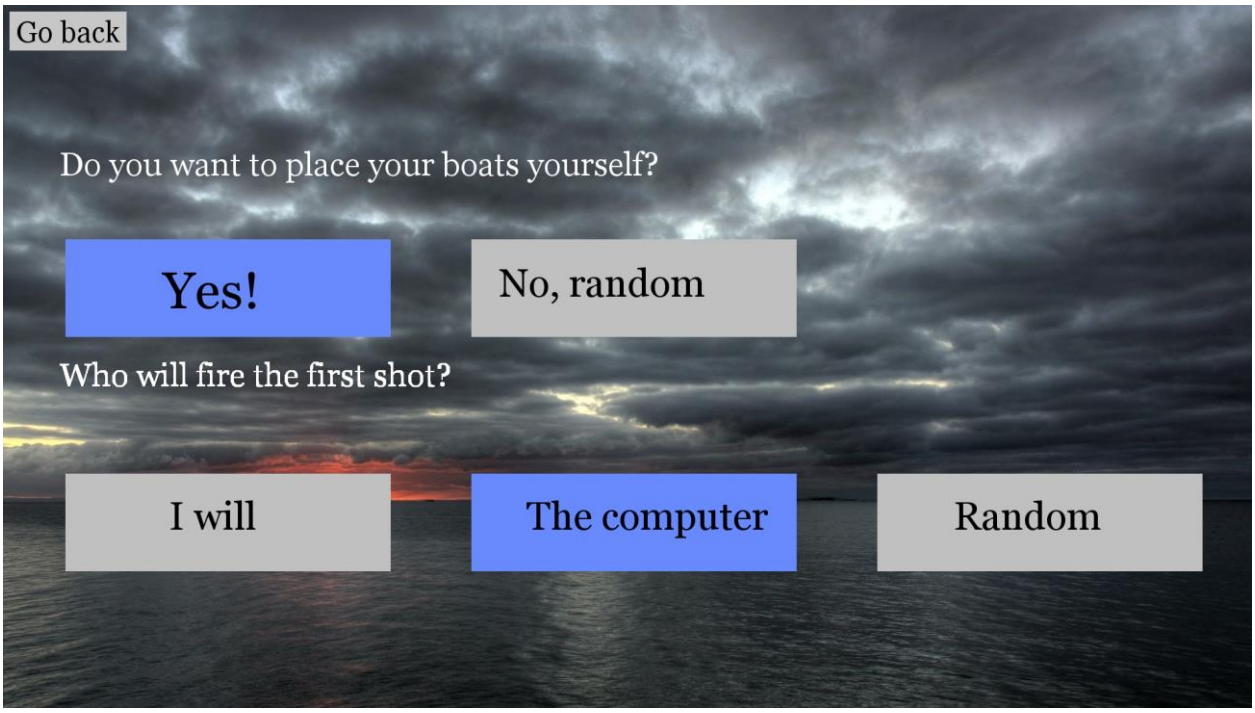
Indien de speler kiest om een nieuw spel te starten, moet hij eerst een aantal opties instellen (zie afbeelding 2 tot en met afbeelding 4). Als eerste wordt de moeilijkheidsgraad ingesteld, dit bepaalt welke denkwijze de computer zal gebruiken tijdens het spel. Dit wordt verder uitgelegd in het deel “Algoritme”.

Afb. 2: Selectiescherm voor de moeilijkheidsgraad

Om de variatie van het spel te vergroten, zijn er meerdere bootpakketten toegevoegd. Zo kan de speler kiezen met welk type boten het spel zal gespeeld worden. In elk pakket zijn de lengtes van de boten verschillend.

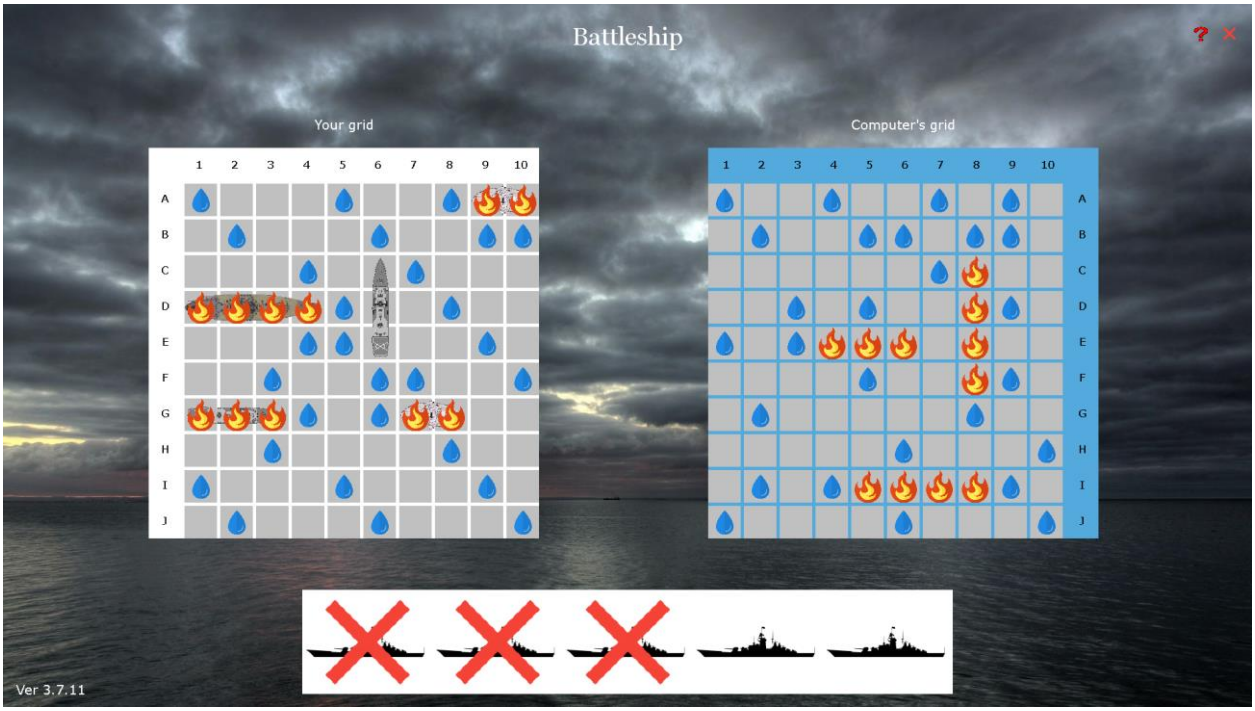
Afb. 3: Selectiescherm voor het botenpakket

Na het kiezen van het botenpakket krijgt men de optie om de boten zelf te plaatsen of deze random in het speelveld te laten plaatsen. Ook mag men beslissen wie de eerste zet zal uitvoeren. Indien men niet kan kiezen, is er ook de optie om het spel te laten beslissen wie er begint. Dit gebeurt via de 'Random' toets.



Afb. 4: Selectiescherm voor de opties

Hierna begint het spel. De speler plaatst, indien hij voor die optie koos, de boten op het speelveld en begint nadien als het zijn beurt is met schieten op het speelveld. Zo wordt er elk om de beurt geschoten tot wanneer de speler/computer heeft gewonnen of de gebruiker het spel heeft afgesloten. In afbeelding 5 wordt weergegeven hoe het spel eruit ziet. Wanneer een schot niet raak was, dan wordt een druppel afgebeeld in de corresponderende cel. In het andere geval zal een vlam getoond worden in die cel.



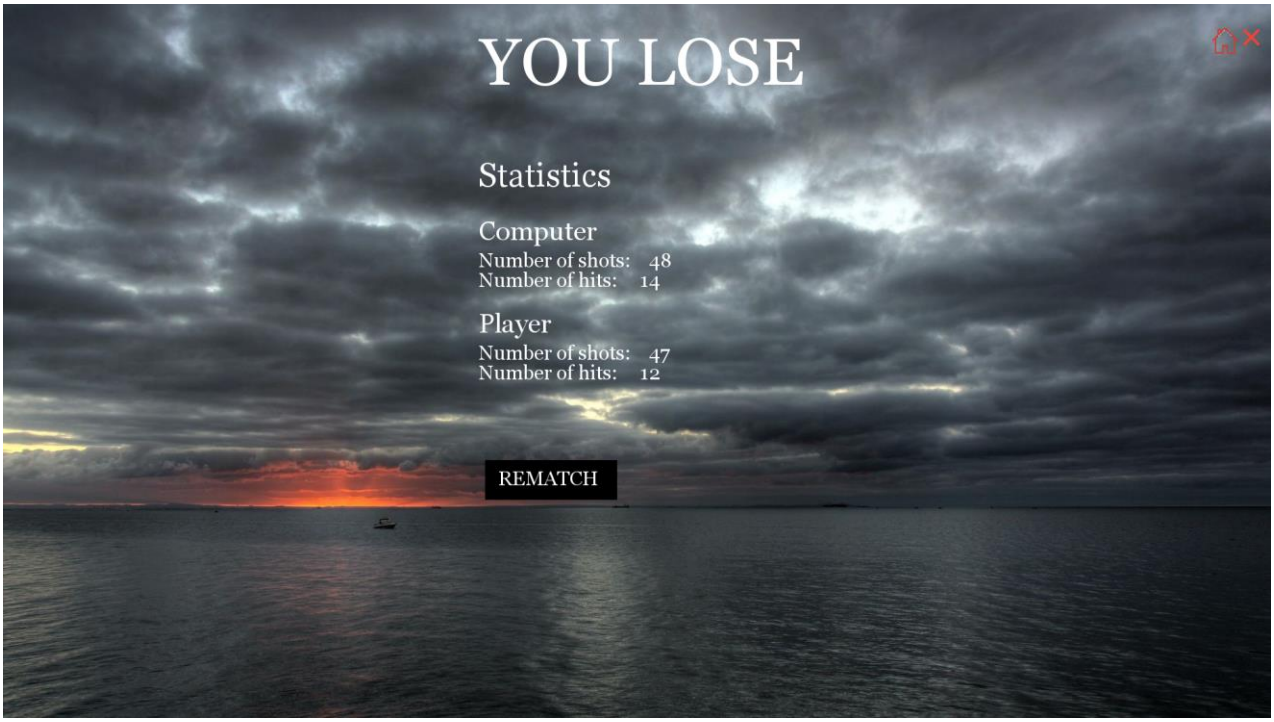
Afb. 5: Het spel

Wanneer de gebruiker het spel afsluit, krijgt hij de optie om het spel op te slaan onder een zelf te kiezen naam. Nadien kan hij het spel dan verderzetten door het savebestand met de opgegeven naam in te laden. Dit wordt weergegeven in afbeelding 6 en 7. Zo heeft elke gebruiker als het ware zijn eigen profiel. In de sectie “Save/laad-functie” wordt besproken hoe dit gerealiseerd is.

Wanneer de speler of de computer alle boten van de tegenstander heeft geraakt, zal het spel de eindresultaten tonen. In dit scherm is er ook de optie om opnieuw een spel te spelen, waarbij dan eerst weer alle opties moeten ingesteld worden. Er kan ook teruggekeerd worden naar het home-scherm of het spel kan ook afgesloten worden. Dit scherm wordt getoond in afbeelding 8.

Afb. 6: Het afsluitscherm

Afb. 7: Het savescherm



Afb. 8: Het eindscherm

Analyse

Klassendiagram

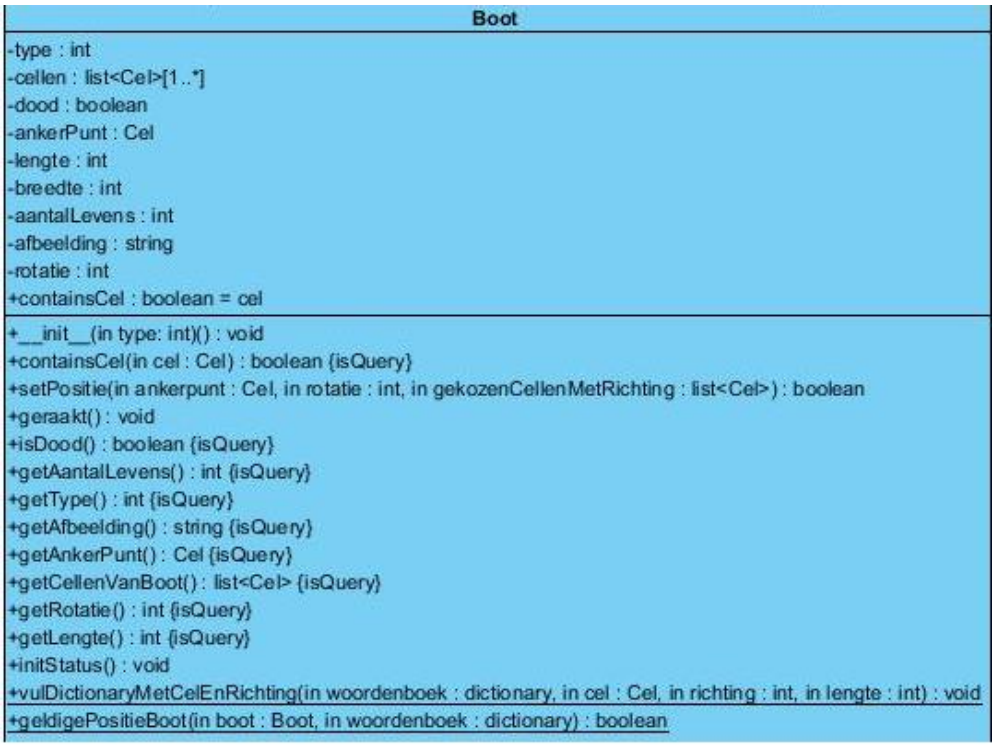
In het project werd gebruik gemaakt van verschillende klassen. Deze klassen kunnen opgedeeld worden in 2 grote groepen, namelijk de grafische klassen en de niet-grafische klassen. De grafische klassen (= view) regelen alles wat er te zien is in de applicatie en de niet-grafische klassen (= model) regelen alle berekeningen die in de achtergrond worden uitgevoerd. Tot de grafische klassen behoren de hulpklasse, het speelveld en het menu. Alle andere klassen kunnen tot de niet-grafische klassen gerekend worden. Nu wordt een korte toelichting gegeven van de functie van elke klasse.

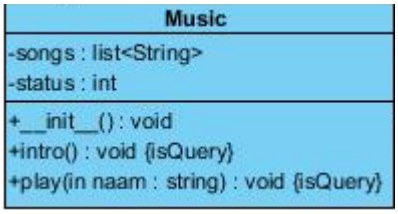
In de klasse Cel wordt de informatie bijgehouden of er op dit vakje al geschoten is en of er een boot opstaat. Deze klasse bevat ook andere handige informatie waaronder bijvoorbeeld de coördinaten van elke cel in de grid. De klasse Grid bevat een verzameling van alle cellen waarmee later gespeeld zal worden. De Grid houdt ook bij waar de objecten van de klasse Boot zich bevinden. Die staan namelijk gepositioneerd op de grid. Een object van de klasse Boot bestaat uit een ankercel, een type en een richting. De ankercel is het beginpunt van de boot en het type duidt op de afbeelding die zal gebruikt worden om deze boot voor te stellen. De richting bepaalt of de boot bijvoorbeeld naar onder wijst. Later tijdens het spel zal het de bedoeling zijn dat de speler de boten van zijn tegenstander zal laten zinken door op de juiste cellen (de cellen waaruit de boot bestaat) te klikken. Dit gebeurt vanuit de klasse Speler of vanuit de klasse Computer. Via de klasse Speler kan een fysieke speler het spel spelen tegen de zeeslag-bot. Deze wordt bestuurd met behulp van de computer. De gebruiker kan het spel makkelijk gebruiken

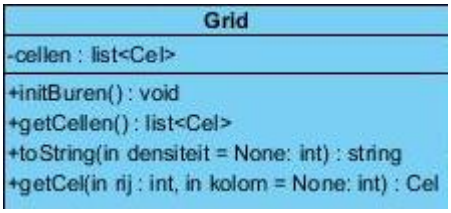
door de klasse Menu. Hierdoor wordt de gebruiker in staat gesteld om makkelijk alle functionaliteiten van het spel te kunnen gebruiken.
De klassen interageren met elkaar via verschillende relaties. In deze paragraaf worden de verschillende relaties tussen deze klassen besproken. De eerste relatie die besproken wordt is de relatie tussen de klasse Boot en de klasse Cel. Deze 2 klassen worden verbonden door een aggregatie: de klasse Boot kan niet bestaan zonder de klasse Cel. De klasse Cel is dan weer via een compositie verbonden met de klasse Grid. De klasse Cel heeft geen nut als de klasse Grid niet bestaat en omgekeerd. De klasse Grid is dan weer via aggregaties verbonden met de klassen Speler en Computer. De grid kan op zichzelf bestaan, maar zonder grid heb je niets aan de klassen Speler en de klasse Computer. De grid kan ook 1 of meerdere boten bevatten.

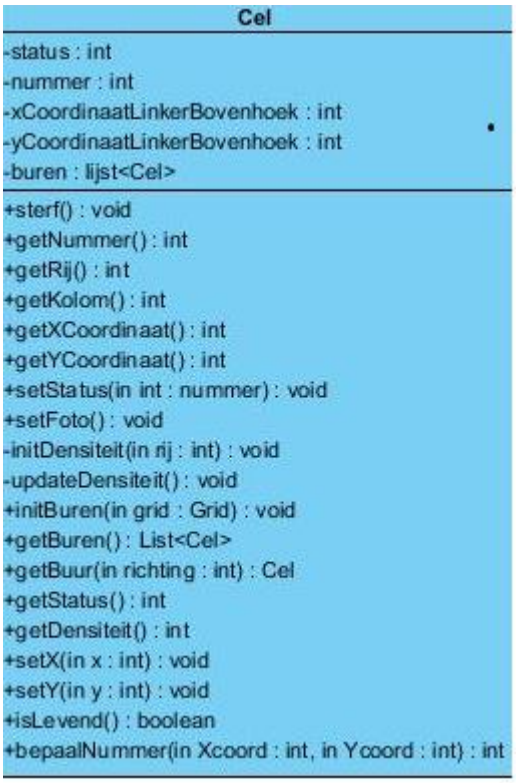
De grafische klassen interageren ook met elkaar via verschillende relaties. De Hulpklasse wordt gebruikt door zowel de klasse Speelveld als de klasse Menu. Door de Hulpklasse wordt er namelijk duplicated code vermeden in deze klassen. De klassen Menu en Speelveld zijn verbonden via een aggregatie. Via deze relatie zijn de grafische klassen en de niet-grafische klassen met elkaar verbonden. De klassen Menu en Speelveld bevatten ook allebei knoppen. Deze zijn verbonden met de klasse Knop via een associatie. De klasse Menu maakt gebruikt van de klasse Music en de klasse SaveLoad. Dit om respectievelijk de muziek en het opslaan van het spel te regelen. Ook de klasse Speelveld maakt gebruik van de klasse Music om muziek te kunnen afspelen

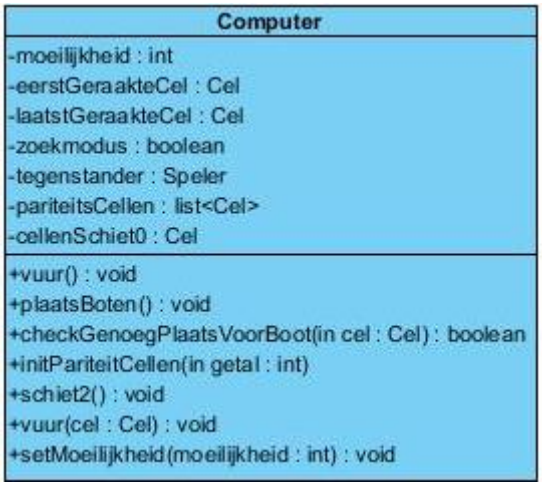
Hieronder wordt elke klasse apart getoond met zijn methodes (al deze klassen vormen samen afbeelding 9). Daarna wordt het lege klassendiagram getoond (zie afbeelding 10). De reden waarom we het klassendiagram opgesplitst hebben, is omdat het op deze manier makkelijk kan ingevoegd worden in het verslag.

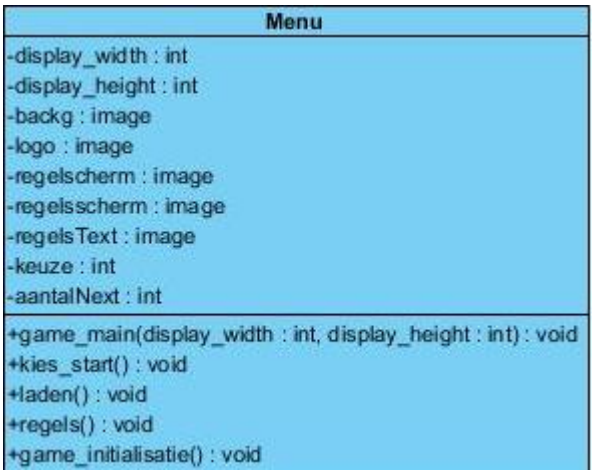


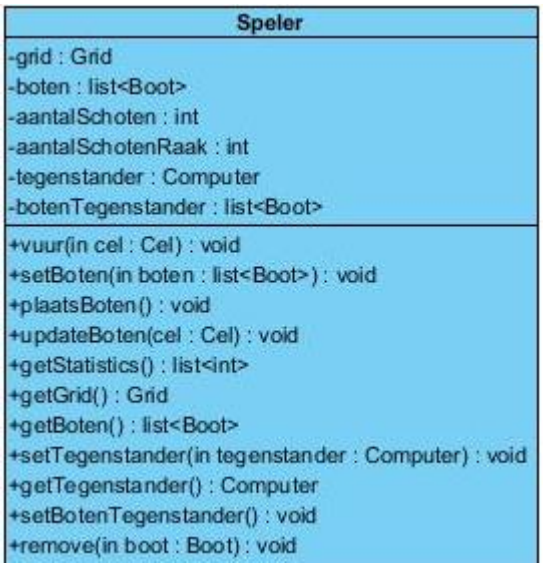

Music
-songs : list<String>
-status : int
+__init__() : void
+intro() : void {isQuery}
+play(in naam : string) : void {isQuery}


Grid
-cellen : list<Cel>
+initBuren() : void
+getCellen() : list<Cel>
+toString(in densiteit = None: int) : string
+getCel(in rij : int, in kolom = None: int) : Cel

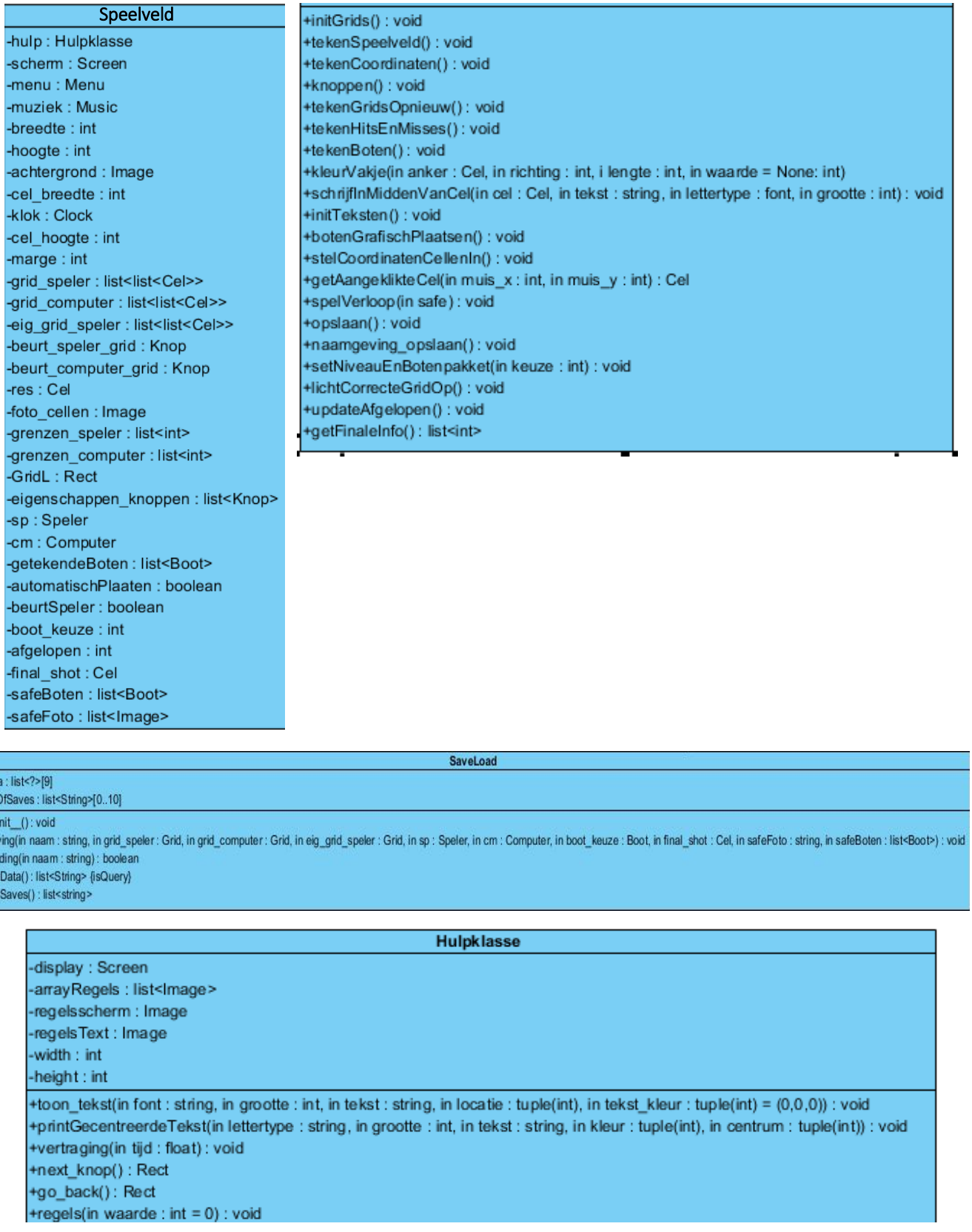

Cel
-status : int
-nummer : int
-xCoordinaatLinkerBovenhoek : int
-yCoordinaatLinkerBovenhoek : int
-buren : lijst<Cel>
+sterf() : void
+getNummer() : int
+getRij() : int
+getKolom() : int
+getXCoordinaat() : int
+getYCoordinaat() : int
+setStatus(in int : nummer) : void
+setFoto() : void
-initDensiteit(in rij : int) : void
-updateDensiteit() : void
+initBuren(in grid : Grid) : void
+getBuren() : List<Cel>
+getBuur(in richting : int) : Cel
+getStatus() : int
+getDensiteit() : int
+setX(in x : int) : void
+setY(in y : int) : void
+isLevend() : boolean
+bepaalNummer(in Xcoord : int, in Ycoord : int) : int


Computer
-moeilijkheid : int
-eerstGeraakteCel : Cel
-laatstGeraakteCel : Cel
-zoekmodus : boolean
-tegenstander : Speler
-pariteitsCellen : list<Cel>
-cellenSchiet0 : Cel
+vuur() : void
+plaatsBoten() : void
+checkGenoegPlaatsVoorBoot(in cel : Cel) : boolean
+initPariteitCellen(in getal : int)
+schiet2() : void
+vuur(cel : Cel) : void
+setMoeilijkheid(moeilijkheid : int) : void

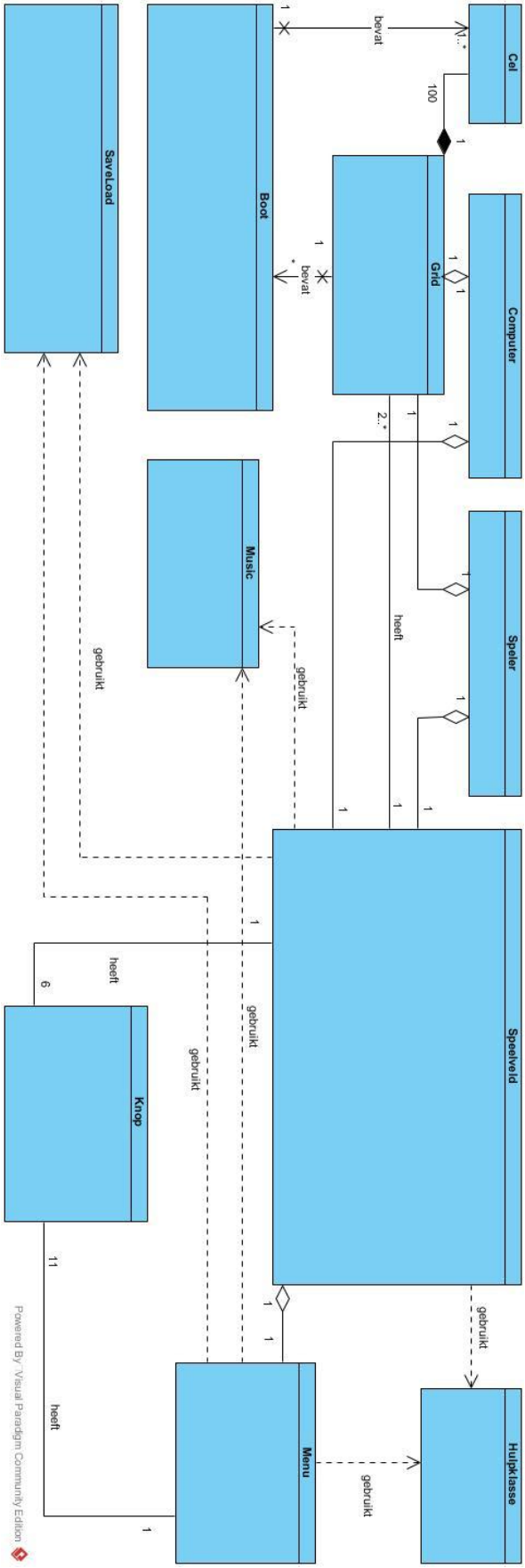

Menu
-display_width : int
-display_height : int
-backg : image
-logo : image
-regelscherm : image
-regelsscherm : image
-regelsText : image
-keuze : int
-aantalNext : int
+game_main(display_width : int, display_height : int) : void
+kies_start() : void
+laden() : void
+regels() : void
+game_initialisatie() : void


Speler
-grid : Grid
-boten : list<Boot>
-aantalSchoten : int
-aantalSchotenRaak : int
-tegenstander : Computer
-botenTegenstander : list<Boot>
+vuur(in cel : Cel) : void
+setBoten(in boten : list<Boot>) : void
+plaatsBoten() : void
+updateBoten(cel : Cel) : void
+getStatistics() : list<int>
+getGrid() : Grid
+getBoten() : list<Boot>
+setTegenstander(in tegenstander : Computer) : void
+getTegenstander() : Computer
+setBotenTegenstander() : void
+remove(in boot : Boot) : void


Knop
-kleur : Color
-hover : boolean
-geklikt : boolen
-enabled : boolean
-fouteBoot : boolean
-achtergrond : boolean
+get_kleur() : Color
+set_hover(in waarde : boolean) : void
+set_selected(in waarde : boolean) : void
+set_achtergrond(in waarde : boolean) : void
+get_selected() : boolean
+set_enabled(in boolean : waarde) : void
+setFouteBoot(in boolean waarde) : void



Afb. 9: Klassen met hun attributen en methodes



Afb. 10: Klassendiagram

Tekstuele use case

Naam
Zeeslag

Doelstelling
Zeeslag spelen tegen de computer

Actoren
- Speler

Precondities
- De speler selecteerde start in het hoofdmenu.
- De speler bepaalde zijn voorkeuren, bijvoorbeeld: moeilijkheid, welke types boten, …
- Het spel werd gestart.

Postcondities
- Het spel is afgelopen en er is een winnaar.

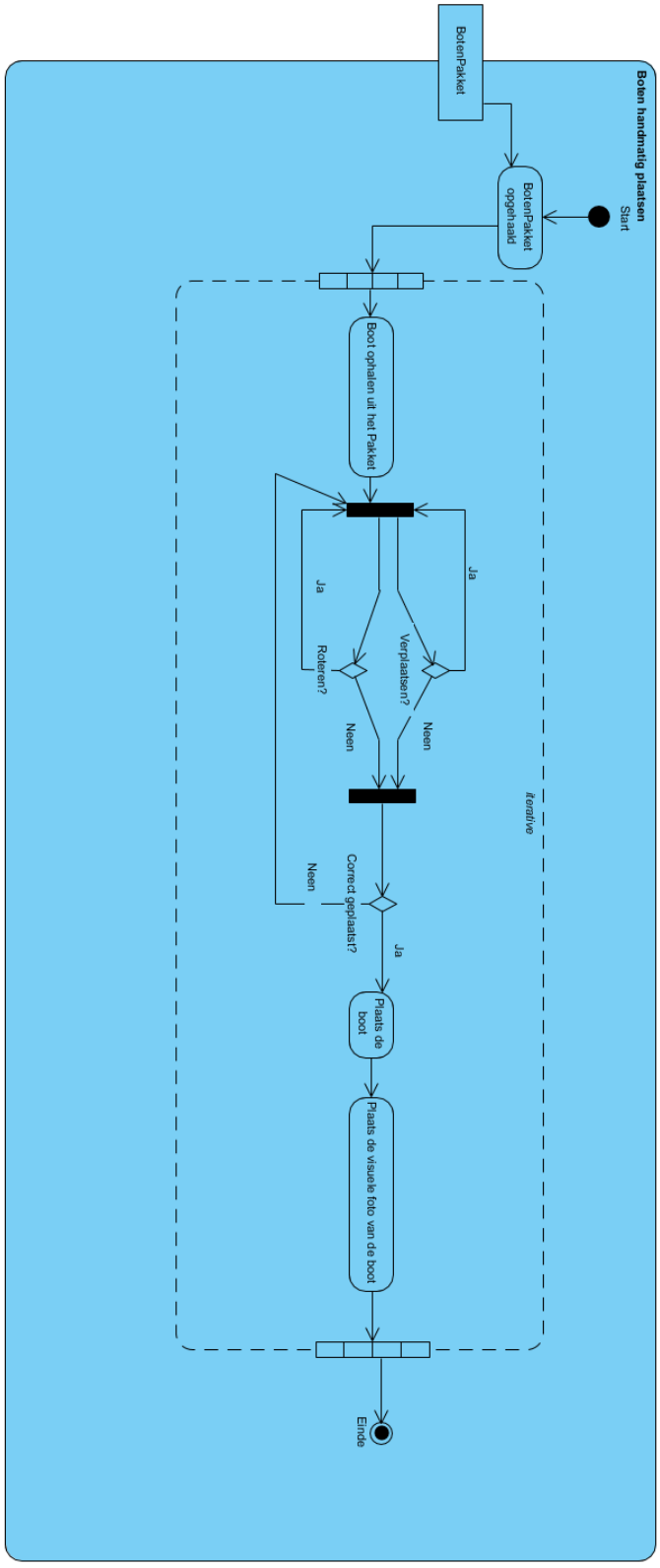
Successcenario
1. De speler/computer is aan de beurt en vuurt.
2. De speler/computer mist of raakt een boot.
3. De speler/computer heeft nog niet gewonnen.
4. De beurt gaat nu over naar de andere.

Alternatieve scenario's
2.a Door het schot van de speler/computer is de boot gezonken, ga naar stap 3.
3.a De speler/computer heeft gewonnen want alle boten van de tegenstander zijn verwoest. Het spel stopt.

Bij dit diagram hebben we gekozen om een tekstuele use case te maken. Het verloop van het spel kan niet ideaal weergegeven worden aan de hand van actoren en afbeeldingen. In een gezelschapsspel is er ook altijd een duidelijke herhaling (lees: while-lus) en hiervoor is het standaard diagram niet geschikt. Wanneer de applicatie start, heeft de speler een aantal keuzemogelijkheden. Hij kan op start drukken waarna er een scherm wordt getoond met opties zoals moeilijkheid, type boten, … Na alles ingesteld te hebben, kan het spel van start gaan. Deze stappen vormen dan ook de precondities. In het hoofdmenu kan de speler ook de regels/besturing bekijken en eventueel het spel herladen maar dit is minder van belang in dit diagram.

Als de speler aan de beurt is, kiest hij om een bepaalde cel aan te vallen. Hierbij zijn er verschillende ontknopingen mogelijk. Ofwel mist de speler (hij raakt dus geen enkele boot) en dan is de computer aan de beurt. Een boot kan ook geraakt worden en deze zal al dan niet zinken, afhankelijk van het aantal reeds geraakte cellen. Een boot zal pas zinken wanneer al de cellen, waarover hij zich uitspreidt, geraakt zijn. Dit betekent niet dat de speler al gewonnen is. De speler wint pas wanneer alle boten van de computer gezonken zijn. De computer doorloopt eigenlijk een gelijkaardig scenario maar deze kiest dan op basis van het algoritme welke cellen het aanvalt

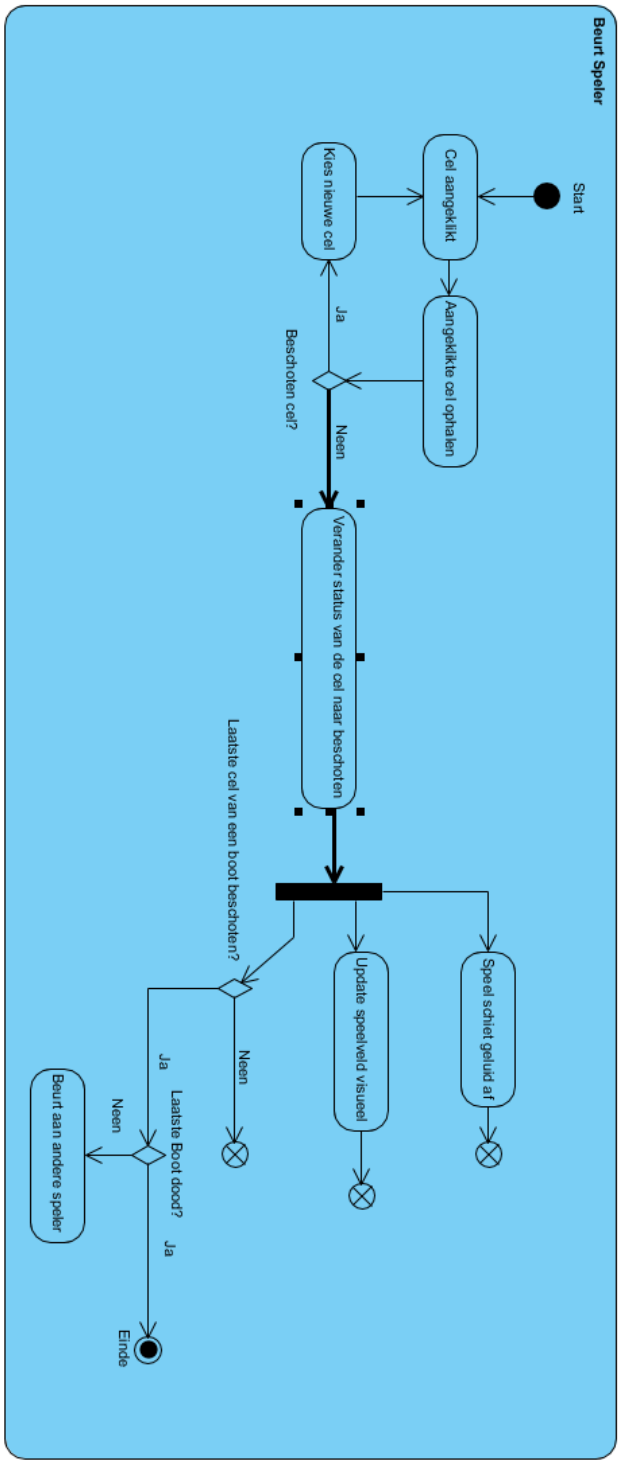
Activiteitendiagram boten plaatsen



Afb. 11: Activiteitendiagram

Indien de speler beslist om de boten handmatig te plaatsen, wordt voor elke boot in het botenpakket (dat de 5 gekozen boten bevat die moeten geplaatst worden) volgende stappen overlopen. De eerste boot uit het pakket wordt opgehaald, nadien krijgt de speler de keuze om te boot te verplaatsen en te roteren totdat de boot zich op de gewenste locatie bevindt. Tijdens dit proces wordt de boot per rotatie/verplaatsing visueel geüpdatet. Bij het plaatsen van de boot wordt gecontroleerd of de boot zich op een correcte locatie bevindt. Indien dit niet het geval is krijgt de speler opnieuw de kans om de boot te roteren en te verplaatsen. Wanneer de boot op een correcte locatie wordt geplaatst, zal het speelveld de locatie van de boot onthouden en visueel de boot op deze locatie afbeelden. Nadien wordt de volgende boot opgehaald en herhalen bovenstaande stappen zich totdat alle boten uit het botenpakket geplaatst zijn. Hierna zal het spel beginnen en kan er op het speelveld geschoten worden. Het activiteitendiagram hiervan wordt weergegeven in afbeelding 11 (vorige pagina).

Activiteitendiagram beurt speler



Afb. 12: Het activiteitendiagram

De speler zijn beurt begint met het aanklikken van een cel op het speelveld van de computer, dit is het rechtse speelveld. Hierna wordt er gecontroleerd of de aangeklikte cel een cel is die nog niet is beschoten. Indien dit wel het geval is, gebeurt er niks en moet de speler een andere cel aanklikken. Indien de aangeklikte cel nog niet beschoten is, wordt de status van de cel aangepast en zal er een schiet geluid afspelen. Indien er zich geen boot op de beschoten locatie bevindt, komt er een druppel tevoorschijn op het speelveld en gaat de beurt naar de computer. Als er zich echter een boot op de beschoten locatie bevindt, komt er een vlam tevoorschijn en controleert men of de laatste cel van een boot is geraakt. Indien het de laatste cel van de laatste boot was, zal het spel afsluiten en heb je gewonnen. Anders gaat de beurt naar de computer.

Sequentiediagram

Het algemeen verloop

Het sequentiediagram (zie afbeelding 13) gaat over het algemene verloop bij het spelen van een spel. Er wordt enkel besproken wat er gebeurt als de gebruiker een nieuw spel start. Een spel laden wordt hier dus genegeerd.
De persoon zal vier keer een keuze moeten maken vooraleer het spel start (zie afbeelding 2 tot en met 4). Telkens wanneer een keuze gemaakt wordt, krijgt de gebruiker een volgend scherm te zien. Bij de laatste keuze zal het speelveld aangemaakt worden rekening houdend met de opties van de gebruiker. Tijdens de initialisatie van het speelveld worden ook een object van de klasse Speler aangemaakt en één van de klasse Computer. In de klasse speler wordt vervolgens een grid gemaakt die zijn bijbehorende cellen aanmaakt. Voor beide grids worden de coördinaten ingesteld en dan kan het werkelijke spel beginnen. Vanuit de klasse menu wordt er gekeken of het spel afgelopen is. Deze variabele wordt gewijzigd in de klasse Speelveld van zodra er een winnaar is. Zolang het spel niet afgelopen is, zal de loop zich blijven herhalen. Deze loop wordt in detail besproken in afbeelding 14 met nadruk op het spelverloop.
Als de klasse Menu op de hoogte is gebracht van het einde, vraagt hij de statistieken op. Deze worden getoond in het eindscherm (zie afbeelding 8) wat dan ook stap nummer 18 is in het diagram.

Het spelverloop

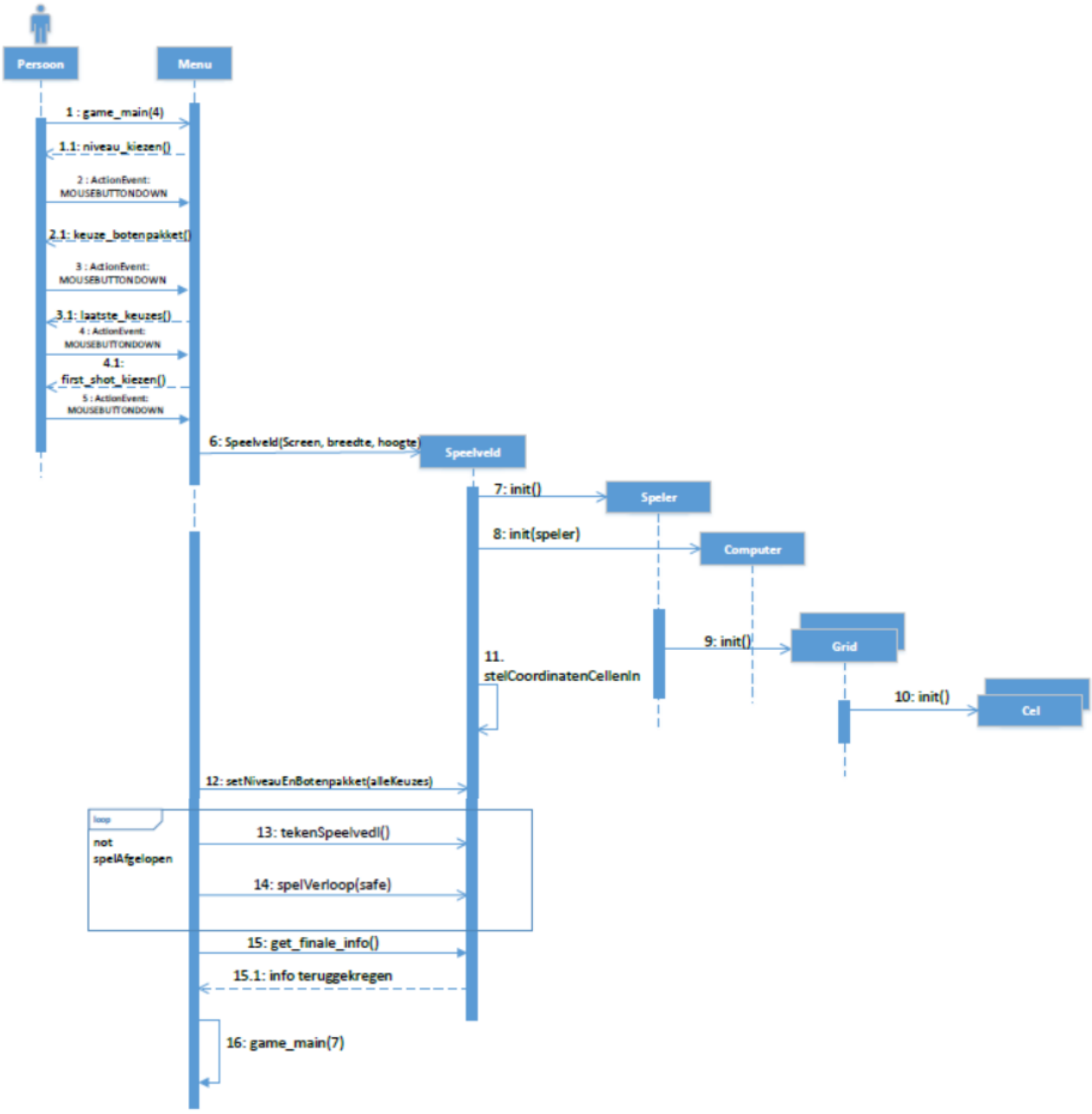
Zolang er geen winnaar is, worden vier methodes opgeroepen. Deze zijn allemaal afkomstig uit de klasse Speelveld. Allereerst wordt de methode tekenSpeelveld() opgeroepen, die zal ervoor zorgen dat alle info grafisch getoond wordt (bijvoorbeeld de boten op de correcte cellen, druppels en vlammen, …).
Vervolgens is er de belangrijkste methode: spelVerloop(safe). Dit is eigenlijk de main van het hele project. De parameter safe heeft betrekking op het feit of er een nieuw spel begonnen werd of een opgeslagen spel.

Nu moeten de boten nog geplaatst worden voor er werkelijk gespeeld kan worden en de parameter safe speelt hierbij een rol. Aangezien een spel niet wordt opgeslagen zonder geplaatste boten, kan hier makkelijk een lijn getrokken worden. Indien er een spel geladen wordt, hoeven de boten alleen nog maar getekend te worden. Heeft de gebruiker gekozen voor een nieuw spel, dan zullen deze nog geplaatst moeten worden. Het plaatsen van de boten is volledig afhankelijk van afbeelding 11. De gebruiker kan

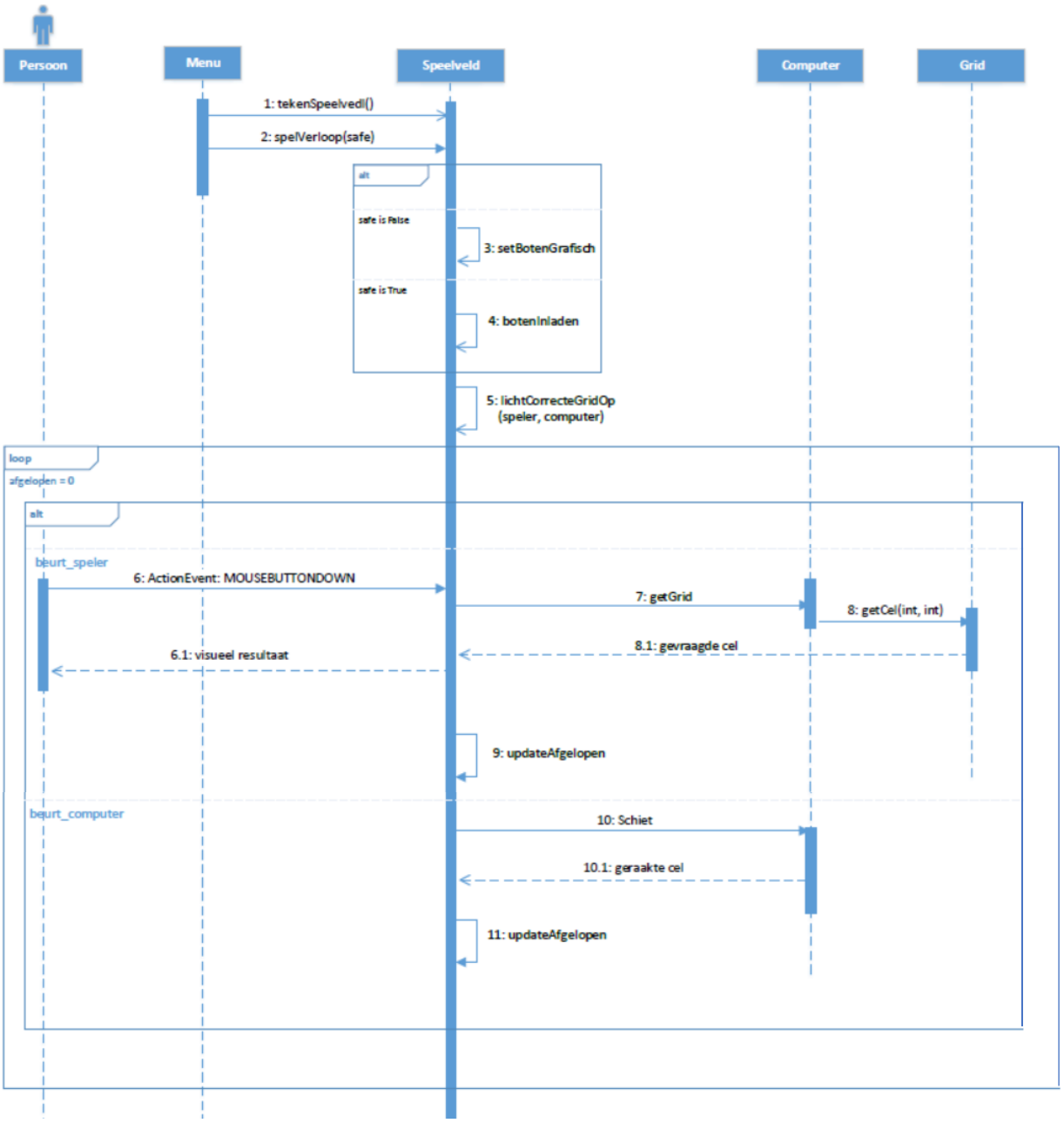
natuurlijk ook gekozen hebben om dit automatisch te laten gebeuren. Dit wordt ook afgehandeld in de methode setBotenGrafisch().

Een kleine methode lichtCorrecteGridOp() toont bij elke beurt wie er mag schieten. Nu is er weer gewerkt met een grote while-lus die verlaten wordt bij een winnaar. Binnenin deze lus wisselt de beurt telkens en schieten de speler en de computer elk om beurt. Stel dat de speler aan de beurt is, dan zal de gebruiker gewoon een cel moeten aanklikken binnen de grid van de computer. Hierbij wordt gecontroleerd of deze cel al eens gekozen werd. Het speelveld zal de bijbehorende grid opvragen en deze zal op zijn beurt de status van de aangeklikte cel ophalen. De verkregen informatie vertaalt zich dan op het scherm. Op het einde van de beurt wordt gecontroleerd of de persoon gewonnen heeft. Indien dit niet het geval is, gaat het spel gewoon verder.

Terwijl de computer aan de beurt is, hoeft de persoon enkel toe te kijken. Het speelveld zal via zijn object Computer de methode schiet() oproepen die via het algoritme een keuze maakt. Die cel wordt dan getoond op het scherm. Hierna wordt opnieuw gekeken of er een winnaar is.



Afb. 13: Sequentiediagram



Afb. 14: Sequentiediagram loop

Algoritme

Begrippen

Om de analyse van het algoritme te kunnen begrijpen, is het belangrijk om enkele begrippen te verduidelijken. Voor nog meer info over het algoritme verwijzen we naar de wiki van GitHub. De link van het algoritme op de wiki verwijst naar de webpagina waarop het gebaseerd is.

- Cel: Het speelveld van zeeslag bestaat uit 100 vakjes, ook wel een cel genoemd. Beide termen duiden op hetzelfde en worden door elkaar gebruikt.
- Pariteit: Stel dat het speelveld van zeeslag ingedeeld is zoals een schaakbord, dus met witte en zwarte vakjes. Dan zal elke boot minstens op één wit en één zwart vakje staan aangezien de kleinste boot twee vakjes lang is. We zien dus dat er bijvoorbeeld enkel op de witte vakjes geschoten moet worden om elke boot te raken. Dit wordt een even pariteit genoemd of een pariteit van twee.
- Densiteit: De densiteit van een cel werkt als een parameter die aanduidt hoe groot de kans is dat er een boot op die cel staat. In het midden van het veld is er meer kans dat er een boot staat dan aan de rand. Een boot kan in het midden namelijk op meerdere manieren gepositioneerd zijn, dit in tegenstelling tot de rand. Als een cel beschoten is, hebben de buurcellen minder kans dat er zich daar een boot bevindt. Het is immers in zeeslag zo dat boten niet evenwijdig naast mekaar mogen liggen.
- Richting: Er worden vier richtingen gebruikt, aangeduid met 0, 1, 2 en 3. Respectievelijk betekent dit: boven, rechts, onder en links. Er wordt ook soms gesproken over zin, dit is om mogelijke verwarring te vermijden met het woord richting dat in wetenschappelijke context bestaat uit twee zinnen.

Analyse algoritme

Het spel heeft drie moeilijkheidsniveaus. Eerst wordt het algoritme van de hoogste moeilijkheid besproken en daarna worden de verschillen met de andere moeilijkheidsniveaus uitgelegd. Het activiteitendiagram is te zien op afbeelding 17.

Zoekmodus

Er wordt een lijst gemaakt van alle cellen die op dat moment de hoogste densiteit hebben. Uiteraard bestaat die lijst enkel uit cellen met pariteit twee. Uit die lijst wordt een willekeurige cel beschoten. Als dat schot een cel met een boot erop heeft geraakt, dan schakelt het algoritme over naar de targetmodus. In deze modus zal de boot volledig zinken. Hierbij worden ook de eerst- en laatstGeraaktCel ingesteld.

Het is echter mogelijk dat de hierboven vermelde lijst leeg is. Dit wil zeggen dat alle pariteitscellen al beschoten zijn. Deze zeldzame situatie treedt op wanneer een boot niet gezonken werd door de targetmodus en de overgebleven cellen van die boot niet op de pariteit liggen. Als oplossing wordt de pariteit op één gezet. Hierdoor worden nu alle cellen in rekening gebracht bij het maken van de lijst van cellen met de hoogste densiteit. Als de zoekmodus een schot vuurt dat geen boot raakt, dan wordt de densiteit van die cel en omliggende cellen geüpdatet.

Targetmodus

Als dit het eerste schot is op een boot, dan is er nog geen schietrichting bepaald. We weten namelijk nog niet of de boot horizontaal of verticaal ligt. De schietrichting wordt dan bepaald door te kijken in welke zin de meeste lege cellen gelegen zijn. Zijn er meerdere richtingen met evenveel lege cellen, dan wordt een willekeurige richting daaruit gekozen om het algoritme minder voorspelbaar te maken. Er bestaat ook een kleine kans dat er een cel geraakt is waarvan alle buurcellen al beschoten zijn. In dit geval zoekt het algoritme naar de zin waar een beschoten cel ligt met een boot en zoekt het in die zin verder tot het een cel tegenkomt die nog niet beschoten is.

Wanneer de schietrichting wel al bepaald is, wordt er gewoon verder in die zin geschoten. Als het schot mist, wordt de zin 180 graden gedraaid. Dit betekent immers dat we voorbij de boot aan het schieten zijn. Er kunnen echter wel enkele speciale gevallen optreden, bijvoorbeeld als de cel in die schietrichting niet bestaat. Hierbij zal de schietrichting nog voor het schot al omgekeerd zijn. Aangezien boten loodrecht naast elkaar kunnen liggen, kan het voorvallen dat de cel, die beschoten moet worden in die schietrichting, al beschoten is en raak was. Het algoritme gaat dan naar de cel erachter kijken tot het een niet beschoten cel vindt.

Als de beschoten cel raak is, dan wordt de schietrichting vastgelegd en wordt de laatstGeraakteCel geüpdatet. Is de boot uiteindelijk volledig beschoten en dus gezonken, dan zal het algoritme terugkeren naar de zoekmodus.

Andere moeilijkheidsniveaus

Het algoritme van gemiddelde moeilijkheidsgraad zal in de zoekmodus compleet willekeurig schieten, zonder rekening te houden met pariteit of densiteit. De targetmodus werkt compleet analoog aan de targetmodus van het hoogste moeilijkheidsniveau.

Het algoritme van makkelijke moeilijkheidsgraad zal in de zoekmodus hetzelfde werken als het gemiddelde algoritme. Er wordt dus ook niet gekeken naar pariteit, densiteit, … In de targetmodus zal het algoritme de eerste buurcel aanvallen die nog niet beschoten is. Eerst zal de bovenste cel aangevallen worden en vervolgens met de klok mee draaien tot de richting is vastgelegd. Op deze manier is het

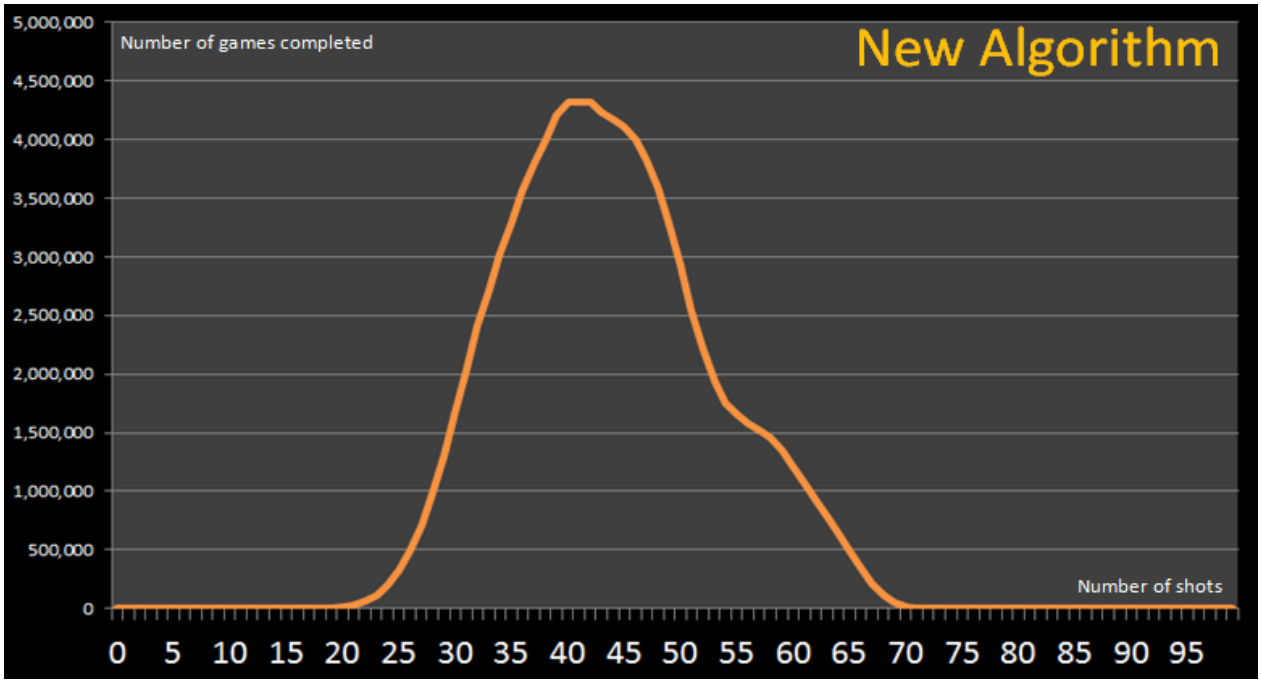
makkelijke algoritme ook veel meer voorspelbaar. De targetmodus zal nog altijd van richting veranderen en trachten de boot te doen zinken. Echter in veel gevallen heeft het algoritme geen oplossing, zoals de andere twee algoritmes wel hebben, en zal het een willekeurig schot afvuren.

Vergelijken met bron

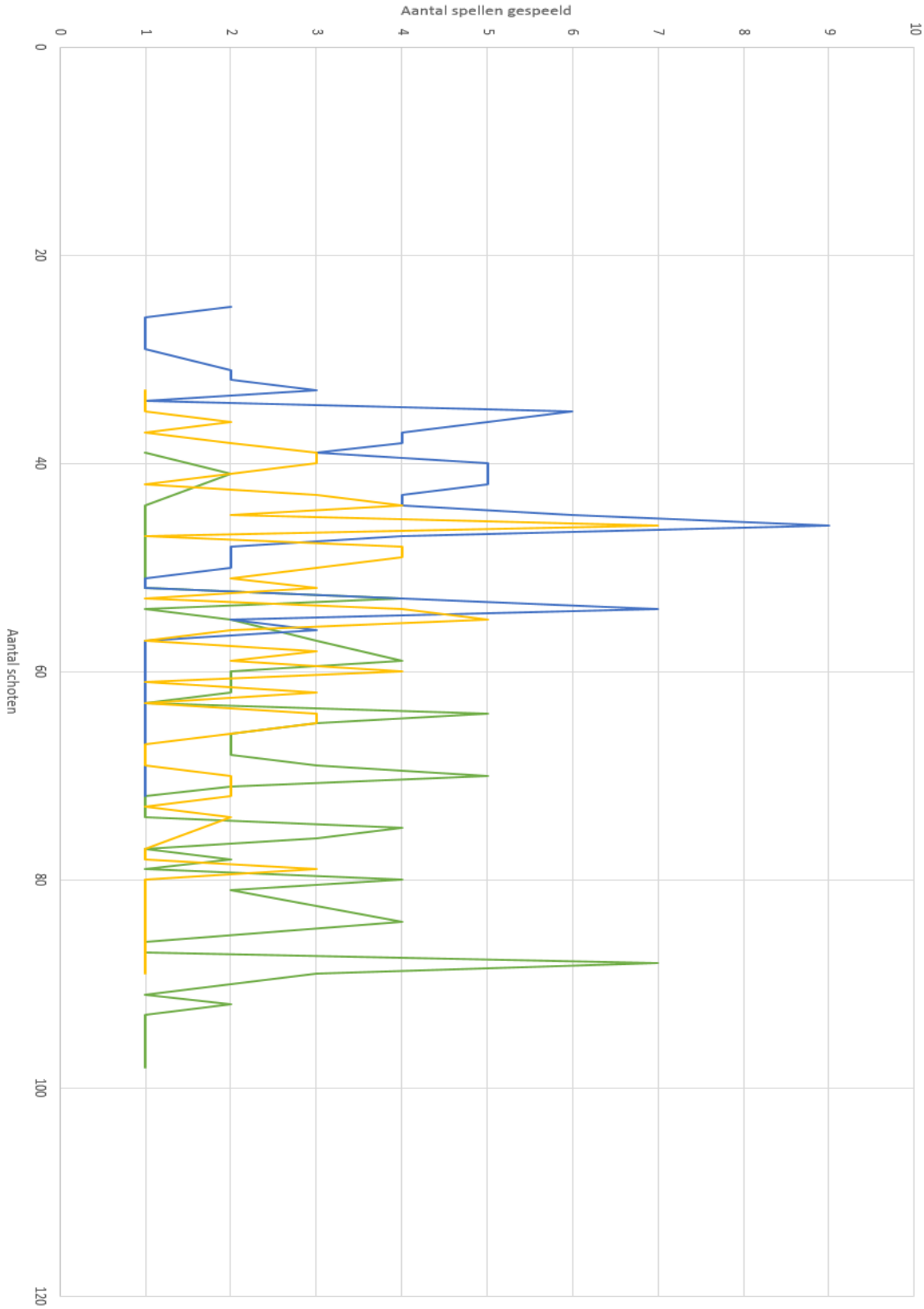
Elk algoritme hebben we 100 keer tegen zichzelf laten spelen. De resultaten daarvan zijn te zien in afbeelding 16. Gemiddeld had het algoritme van de hoogste moeilijkheidsgraad 43,08 zetten nodig. Voor het gemiddeld en makkelijk moeilijkheidsniveau is dit respectievelijk 55,17 en 71,03 zetten. Het algoritme waar we ons op gebaseerd hebben, heeft een gelijkaardige test gedaan. De resultaten van de bron worden getoond in afbeelding 15. Deze afbeelding komt overeen met de blauwe grafiek in afbeelding 16. Beide hebben een gelijkaardige spreiding. Dit betekent dat ons algoritme ongeveer even goed is als dat van de bron documentatie, zeker een geslaagde implementatie dus.

Het is wel belangrijk om te weten dat onze testen en die van de bron documentatie gebruik maakten van dezelfde boten (17 vakjes met een boot). Ook goed om te weten is dat onze testen gebaseerd zijn op 100 pogingen terwijl de test van de bron miljoenen pogingen heeft uitgevoerd. Het is duidelijk dat zoveel pogingen onmogelijk waren voor ons.

De legende voor de grafiek in afbeelding 16 is als volgt: blauw voor het moeilijke algoritme, geel voor het gemiddelde algoritme en groen voor het makkelijke algoritme.



Afb. 15: De resultaten van het oorspronkelijke algoritme



Afb. 16: De resultaten van ons algoritme

Activiteitendiagram Algoritme

Afb. 17: Activiteitendiagram algoritme

Save/laad-functie

Om het spel van een save/laad functie te voorzien, werd er besloten om de pickle module van Python te implementeren. Deze module implementeert binaire protocollen voor het serialiseren en de-serialiseren van verschillende objecten in Python. Elk object wordt hierbij omgezet naar een reeks bytes die dan later makkelijk kan terug gezet worden. Een object kan gepickled worden indien elke variabele van het object kan gepickled worden. In ons project was dit voor de meeste objecten (zoals de computer, de speler, de grids, …) geen probleem. Enkel de lijst van pygame objecten, die de visualisatie van de boten bijhoudt, was niet te pickle'en. Daarom wordt tijdens het plaatsen van de boten hun positie en rotatie onthouden in aparte lijsten (die wel gepickled kunnen worden). Zo kunnen deze bij het inladen automatisch geplaatst worden als pygame objecten. De reeks bytes, die na het pickle'en overblijven, zijn onleesbaar voor mensen waardoor deze niet handmatig aangepast kunnen worden om zo bijvoorbeeld het spel te winnen. Zoals eerder vermeld kan elke gebruiker zijn eigen spel opslaan. Dit wordt gerealiseerd door elke gebruiker zijn save game te op te slaan in een apart bestand. De bestandsnaam is dan de naam die opgegeven is door de gebruiker.

Omzetten naar exe

Om ervoor te zorgen dat een gebruiker makkelijk onze game kan starten, hebben we besloten om een exe bestand aan te maken. Na wat opzoekwerk hebben we gekozen om PyInstaller te gebruiken. PyInstaller is een programma dat een Python project kan omzetten naar 1 enkel exe bestand. Het voordeel is dat je het exe bestand kan customizen en enkele optimalisaties kan instellen. Dit wordt hieronder verder besproken. Een tweede voordeel is dat het ook ondersteuning biedt voor oudere versies van Python en bovendien biedt het standaard ondersteuning voor verschillende packages zoals PyGame, PyQT, ... Het starten van PyInstaller gebeurt wel via de opdrachtprompt. Deze manier van werken is misschien niet optimaal maar er wordt voldoende uitleg gegeven op de site van PyInstaller hoe dit in zijn werk gaat.

Om de omzetting te starten, open je eerst de opdrachtprompt en navigeer je naar de directory waarin alle klassen staan. Vervolgens moet onderstaand commando uitgevoerd worden (dit is een zeer eenvoudig commando en dit gaat ervan uit dat er geen externe data nodig is, zie hieronder):

```
pyinstaller Menu.py
```

Hierbij zal PyInstaller alle imports checken die zich in die klasse bevinden (bijvoorbeeld PyGame of andere klassen uit het project) en deze ook bundelen in de exe. Je hoeft dus telkens maar 1 klasse te vermelden bij de omzetting. De andere klassen worden dan via die imports meegenomen in de exe. Bij voorkeur is deze klasse degene die als eerste moet worden uitgevoerd. In ons project is dit het menu want van hieruit wordt het spel gestart. Dit commando kan echter nog verbeterd worden. Hieronder wordt het uiteindelijke commando getoond en wat uitleg gegeven:

```
set PYTHONOPTIMIZE=1 && pyinstaller --onefile --name "Battleship" --
add-data "img/achtergrond.png;."  --add-data "img/afsluitknop.png;."  -
-add-data "img/back.jpg;."  --add-data "img/home.png;."  --add-data
"img/hulpknop.png;."  --add-data "img/logo2.png;."  --add-data
"img/regelss.jpg;."  --add-data "img/rulesText.png;."  --add-data
"img/stand1.png;." --add-data "img/stand2.png;."  --add-data
"img/stand3.png;."  --add-data "img/boten/boot_beneden.png;."  --add-
data "img/boten/boot1.png;."  --add-data "img/boten/boot2.png;."  --
add-data "img/boten/boot3.png;."  --add-data "img/boten/boot4.png;."  -
-add-data "img/boten/boot5.png;."  --add-data
"img/boten/boot1gedraaid.png;."  --add-data
"img/boten/boot2gedraaid.png;."  --add-data
"img/boten/boot3gedraaid.png;."  --add-data
"img/boten/boot4gedraaid.png;."  --add-data
"img/boten/boot5gedraaid.png;."  --add-data
"img/boten/bootpakket1.png;."  --add-data "img/boten/bootpakket2.png;."
--add-data "img/boten/bootpakket3.png;."  --add-data
"img/cel/mis.png;."  --add-data "img/cel/raak.png;."  --add-data
"Sounds/blub.wav;."  --add-data "Sounds/intro.wav;."  --add-data
"Sounds/schiet.wav;." --add-data "Boot.csv;." --noconsole --
icon="img/icoon.ico" Menu.py
```

De verschillende add-data's vermelden de verschillende externe bestanden waarvan gebruik gemaakt wordt (bijvoorbeeld afbeeldingen, CSV-bestand, …). Via "`--onefile`" stellen we in dat alles moet gebundeld worden in 1 enkel exe bestand. Anders wordt er een directory aangemaakt met verschillende bestanden. Overigens wordt ook de naam ingesteld met behulp van "`--name "Battleship"`" en het exe bestand krijgt ook een aangepast icoon door middel van "`--icon="img/icoon.ico"`". Er zijn ook een aantal extra optimalisaties ingesteld. Door middel van "`--noconsole`" wordt er nooit een opdrachtprompt geopend wanneer we de exe starten. Dit is ook helemaal niet nodig voor de gebruiker. Als laatste optimalisatie hebben we volgende instructie toegevoegd: "`set PYTHONOPTIMIZE=1`". Dit zorgt ervoor dat de .exe iets efficiënter werkt. Eenmaal de omzetting gebeurd is, is er wel een nadeel opgedoken: de gebruiker zou namelijk bij de exe alle bestanden nog eens moeten opslaan (afbeeldingen, CSV-bestand, …). Dit is uiteraard niet gebruiksvriendelijk en hiervoor hebben we een oplossing voorzien.

Nu is het spel gereduceerd tot een exe file, drie mappen en een CSV-file. Om ervoor te zorgen dat de gebruiker minimale inspanning moet leveren om het spel te spelen, werd er gekozen om een installer te maken. Die zal deze bestanden op de juiste locatie plaatsen en een snelkoppeling voorzien. Inno setup is het programma waarmee dit mogelijk is. Door eer script te schrijven zoals te zien is op afbeelding 18 kan men het volledige spel reduceren tot een executable. In het script dat te zien is op afbeelding 18 kan men onder andere de naam van het programma, de versie van het programma, de Publisher, … meegeven.

```
AppId={{889DCF83-1AA0-4217-AB76-E43200593458}
AppName=Battleship
AppVersion=3.7.11
;AppVerName=Battleship 3.7.11
AppPublisher=Team-12
AppPublisherURL=https://github.ugent.be/projectpython18/team-12
AppSupportURL=https://github.ugent.be/projectpython18/team-12
AppUpdatesURL=https://github.ugent.be/projectpython18/team-12
DefaultDirName={pf}\Battleship
DisableProgramGroupPage=yes
OutputDir=D:\robbe\Documents\team-12\Exe
OutputBaseFilename=BattleshipInstaller
Compression=lzma
SolidCompression=yes
SetupIconFile="D:\robbe\Documents\team-12\klassen\img\icoon.ico"

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "D:\robbe\Documents\team-12\Exe\Battleship.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "D:\robbe\Documents\team-12\klassen\img\*"; DestDir: "{app}\img"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "D:\robbe\Documents\team-12\klassen\Saves\*"; DestDir: "{app}\Saves"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "D:\robbe\Documents\team-12\klassen\Sounds\*"; DestDir: "{app}\Sounds"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "D:\robbe\Documents\team-12\klassen\Boot.csv"; DestDir: "{app}"; Flags: ignoreversion
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: "{commonprograms}\Battleship"; Filename: "{app}\Battleship.exe"
Name: "{commondesktop}\Battleship"; Filename: "{app}\Battleship.exe"; Tasks: desktopicon

[Run]
Filename: "{app}\Battleship.exe"; Description: "{cm:LaunchProgram,Battleship}"; Flags: nowait postinstall skipifsilent
```

Afb. 18: Het installer script

Conclusie

Tijdens dit project was de samenwerking binnen de groep uitstekend. Er werden steeds duidelijke afspraken gemaakt en de taken werden goed verdeeld. De realisatie van het project is ook vlot verlopen en wanneer er problemen waren, waren er altijd wel mensen bereid om te helpen. De analyse was voor ons het moeilijkste deel van het project aangezien dit iets compleet nieuws was. Uiteindelijk hebben we het wel tot een goed einde kunnen brengen. Samengevat was het een uitdagend project maar zeker ook een leuk project waarbij we trots zijn op het eindresultaat!