# Eindverslag Monopoly

## Groepsinformatie

Groep 16: Ho Yuet Cheung en Sebastiaan Weytjens
Aantal gewerkte uren: Ho Yuet: 143 uren
Sebastiaan: 132 uren

## Problemen

### Bugs

- Er kan geen spel opgeslagen worden wanneer het bestand nog niet bestaat.
- Wanneer je met de blauwe dobbelsteen een monopoly rolt, wordt de speler soms niet naar een onbeheerde plaat gebracht. De oorzaak hiervan is onduidelijk.
- Wanneer je een spel wilt laden met een kleiner aantal spelers dan het huidige spel, crasht het programma.
- Wanneer je op een 'majority' huizen hebt gebouwd en dan een 'full-monopoly' krijgt, worden de huizen niet meer evenwichtig gebouwd.
- Wanneer je iets moet betalen, wordt er niet gecheckt of je genoeg geld hebt.
- Wanneer je bij elke nieuwe beurt exact 1 building koopt, wordt er geen rekening gehouden met de evenwichtige bouwregel.
- Wanneer een spel wordt ingeladen kunnen er geen eigendommen gekocht worden en kan er geen huur worden betaald. De oorzaak hiervan is een fout opgeloste merge-fout.
-

### Afwijkingen van de spelregels

- Er is een oneindig aantal 'Get-out-of-jail-free'-kaarten.
- Geld wordt niet als briefjes getoond, maar een speler heeft gewoon een hoeveelheid geld als integer. Bij het betalen aan een andere speler worden er dus ook geen briefjes uitgewisseld.

### Onvolledige onderdelen

- Kanskaarten hebben we niet geïmplementeerd. Wanneer je op een kansvakje terechtkomt, wordt er een algemeen fondskaart getrokken.
- Wanneer er een speler failliet gaat door schulden bij de bank worden zijn bezittingen niet geveild.
- De functie 'endGame' kijkt niet na of er nog maar 1 speler overblijft, er zal dus nooit een winnaar zijn.

## De uiteindelijke taakverdeling met eventuele wijzigingen

### Taakverdeling

Ho Yuet:
- BoardView in orde
- PlayerInfoView in orde
- Bewegen van players op het bord
- Board maken
- Bewegen van Player

- Eigenaars kunnen zien
- Gebouwen tonen
- BuyDialog implementeren
- PlayerInfoView properties kunnen zien
- SpaceView maken en implementeren
- Straten kopen
- Straten verkopen
- Bedrijven kopen
- Bedrijven verkopen
- Gebouwen kopen
- Gebouwen verkopen
- PropertyCard maken en implementeren
- StreetSpace maken en implementeren
- CompanySpace maken en implementeren
- StationSpace maken en implementeren
- ChanceCardView implementeren
- Hypotheek dialogen kunnen zien
- Bezittingen in de hypotheek kunnen zien
- Bezitting in Hypotheek zetten
- Hypotheek aflossen
- Spel laden kunnen kiezen
- Spel opslaan kunnen kiezen
- Spel opslaan
- Spel laden

<u>Sebastiaan:</u>
- DiceView worp kunnen zien
- Player constructor maken
- Game maken
- Veranderen van beurten
- Dice werpen
- Gevangenis implementeren
- Geld krijgen langs start
- Betaling van huur kunnen zien (RentDialog)
- Dialogen voor Bus Tickets
- Speed Dice kunnen gebruiken
- Huur betalen
- Bus tickets maken
- Bus tickets gebruiken
- Speed Dice implementeren
- CommunityChestCardView implementeren
- ChanceCardView implementeren
- Veiling kunnen zien en meedoen
- Kaarten bijhouden (gevangenis)
- CommunityChestCard implementeren
- ChanceCard implementeren
- CommunityChestCard trekken en gebruiken
- ChanceCard trekken en gebruiken
- MoneySpace en resterende afgeleide klasses implementeren
- Veilen
- EndGame dialoog kunnen zien
- Players kunnen verliezen

- Winnaar wordt bepaald

Beiden:
- UML maken
- Backlog opstellen en takenverdeling
- Opstellen requirements
- UML fixen
- Testen
- Implementeren wat nog niet geïmplementeerd is
- Bug fix
- Eindverslag
- Slides in orde brengen
- Presentatie voorbereiden

**Wijzigingen**

- Gevangenis ging Ho Yuet normaal gezien implementeren, maar hebben we onderling gewisseld voor het implementeren van de verkoop en de aankoop van gebouwen, wat Sebastiaan ging voorzien. Dit hebben we gedaan omdat Ho Yuet de gebouwen nodig had om hypotheek verder uit te werken terwijl Sebastiaan nog problemen had met het betalen van huur.
- De initiële taakverdeling gaf aan dat Ho Yuet de klasse "StreetView" zou invoeren, maar dit is veranderd in een klasse "SpaceView", een klasse die alle spaces afhandelt. We hebben dit gedaan omdat de klasse "StreetView" gelijkaardig was aan andere "SpaceViews" in het begin.
- Ho Yuet zou de klasse "StreetSpace" maken, maar dit is opgesplitst in een klasse "StreetSpace" en een klasse "StationSpace". We dachten eerst dat "StationSpace" gelijkaardig was aan "StreetSpace" en de kleur alleen anders zou zijn, maar uiteindelijk bleken er teveel verschillen te zijn tussen een straat en een station.
- We hadden gepland dat Ho Yuet de kanskaarten zou implementeren, maar dit heeft Sebastiaan overgenomen. De kanskaarten zijn ongeveer hetzelfde als de algemeen fondskaarten en om verschillende werkwijzes te vermijden, was het handiger als Sebastiaan dit deed.

## Wijzigingen ten opzichte van het initiële klassendiagram

- Alle subklassen van "MoneySpace", behalve "StartSpace", hebben we verwijderd omdat ze allemaal exact hetzelfde doel hadden (geld geven of vragen aan/van een speler) en dus in één klasse samengevoegd konden worden, namelijk "MoneySpace", afgeleid van "SpecialSpace". "StartSpace" hadden we wel nog nodig en is nu een subklasse van "SpecialSpace" in de plaats van "MoneySpace".
- Om straten, bedrijven en stations voor te stellen hadden we initieel twee klassen die als basisklasse "Space" hadden. "StreetSpace" om de straten en stations voor te stellen in de model en "CompanySpace" om de bedrijven voor te stellen in de model. Dit hebben we veranderd naar "StreetSpace", "StationSpace" en "CompanySpace" die als basisklasse "Space" hebben omdat er toch een aantal verschillen waren tussen straten en stations.
- Huizen, wolkenkrabbers, hotels en raildepots werden voorgesteld door vier verschillende klassen ("HouseView", "HotelView", "SkyScraperView" en "RailDepotView") die afgeleid waren van de klasse "BuildingView". Deze klassen verschilden niet van elkaar en uiteindelijk konden we alle gebouwen voorstellen met de klasse "BuildingView" door alle objecten een bepaalde kleur te geven die het type gebouw aangaf.
- Initieel hadden we "BusCardView", "ChanceCardView" en "CommunityChestCardView", afgeleid van "CardView", om alle kaarten in het spel voor te stellen in de view. We hebben deze klassen vervangen door eenvoudige Messageboxen te tonen in de view wanneer er een kaart gebruikt/ getrokken wordt.
- We hebben een klasse "AuctionSpace", afgeleid van "Space", toegevoegd om het vakje 'Auction any unowned property' voor te stellen in de model.
- De klasse "StreetView", met superklasse "SpaceView" hebben we ook verwijderd omdat we maar 1 klasse nodig hadden om alle verschillende vakjes voor te stellen in de view.

- De klassen "CommunityChestCard" en "ChanceCard" hebben we abstract gemaakt en voor elke soort kaart (Betaal/ontvang geld, betaal aan elke speler, betaal voor elk huis en elk hotel, gratis uit de gevangenis, ga naar start en ga naar de gevangenis) een aparte afgeleide klasse gemaakt. De kanskaarten zijn hetzelfde als de algemeen fondskaarten zoals we eerder al hadden aangegeven.

# Zelfreflectie

## Polymorfisme

In het begin gingen we bij elke beweging van een speler na op welk type vakje de speler geland was d.m.v. een enum "Type" en een functie "getType" waarna de functie van dat soort vakje opgeroepen werd. Dit was echter helemaal tegen het principe van polymorfisme in. Daarom roepen we nu na elke beweging van een speler een functie "doSpaceMove" die a.d.h.v. polymorfisme de juiste functie van het vakje aanroept zodat er bijvoorbeeld huur betaald wordt, een dialoog om te kopen wordt geopend,... We geven bij deze functie ook het bord door zodat we in de functie toegang hebben tot de functies van het bord. Wanneer er toch gecheckt wordt welk type een vakje of een kaart heeft, is dat bedoeld om het loaden en saven goed te laten verlopen.

## High-cohesion en low-coupling

We hebben 'high-cohesion' en 'low-coupling' ook goed proberen toe te passen. We hebben gezorgd dat alle klassen één doel hebben.
In de model hebben we bijvoorbeeld de klassen "Dice" en "SpeedDice" gemaakt om het rollen met de dobbelstenen af te handelen i.p.v. dit te doen binnen de klasse board.
Daarnaast hebben we ook nog de klasse "Player" die alle taken van de speler uitvoert (het betalen/ontvangen van geld, het bewegen) en alle informatie over een specifieke speler bijhoudt (saldo, huidige positie, of een speler in de gevangenis zit of niet).
In de view hebben we dit ook proberen te implementeren. We hadden alle dialogen kunnen weergeven vanuit de klasse "BoardView", maar we hebben voor elke aparte dialoog (voor het kopen van eigendommen, het veilen van eigendommen) aparte klassen gemaakt waarin we alles functies uitvoeren die nodig zijn en alle variabelen bijhouden.
Anderzijds hebben we 1 klasse "BuildableBuildingsView" die het kopen en het verkopen van gebouwen afhandelt. We hadden deze klasse misschien beter in 2 klassen opgesplitst, maar naar onze mening waren er te weinig verschillen tussen de 2 klassen waardoor er veel gekopieerde code zou zijn.
De functie 'checkStart' had misschien beter in de klasse "Player" dan in "Board" gezeten, aangezien de speler de huidige positie bijhoudt en deze nu dus in "Board" opgevraagd moet worden. Anderzijds zouden we de oude positie toch door moeten geven aan 'checkStart' in de klasse "Player" als de functie daarin zat. Daarom hebben we toch gekozen voor "Board".
Daarnaast had 'checkDoublesThrice' ook beter in "Player" dan in "Board" gezeten omdat deze de laatste rolls bijhoudt. Maar we hebben ervoor gekozen om het toch in "Board" te doen aangezien dat beter overeenkwam met onze denkwijze.
We hebben de verschillende delen van een functie proberen op te splitsen zodat deze niet te lang en onduidelijk werd en zodat deze maar 1 doel had. Bij sommige functies had dit misschien beter gekund, bijvoorbeeld hadden we de constructor van "BoardView" beter opgesplitst in verschillende functies voor het initialiseren van de vakjes, het connecteren van de verschillende signals en slots, ... In de klasse "Board" hebben we dit dan weer wel gedaan voor het uitlezen van de vakjes en het uitlezen van de kaarten zodat de constructor overzichtelijk en duidelijk bleef.
Met betrekking tot high-cohesion hebben we de klasse Board die alle modelklassen bevat. Sommige klassen gebruiken dan Board ook zodat ze bepaalde functies kunnen uitvoeren, bijvoorbeeld in Space bestaat er een doSpaceMove functie die gebruikt maakt van polymorfisme en een parameter Board& board heeft dat nodig is voor zijn afgeleide klassen.

## Information expert

Onze 'information-expert' is de klasse "Board". Het bord houdt bijvoorbeeld alle spelers, alle vakjes, alle kans- en algemeen fondskaarten, de speler die aan de beurt is,... bij. De meeste klassen worden dus ook ge‹nstantieerd in het bord waardoor bijna alle spelregels ook worden afgehandeld binnen deze klasse. We hadden deze regels ook kunnen afhandelen in "Game", maar we vonden het bord hier de meest logische keuze. Hierdoor is het bord ook ineens de 'creator' aangezien hierin bijna alle klassen aangemaakt worden.

## Model-View

We hebben een klasse "BoardView" en een klasse "Board" gemaakt. Hierin werden de meeste signals ontvangen en gestuurd om te communiceren tussen de model en de view. We hebben nergens in de model functies van de view aangeroepen en in de view hebben we enkel een instantie van "Game" aangemaakt waarop enkele functies zoals 'getBoard' worden aangeroepen. Dit deden we vooral om de model met de view te connecteren.

## Public, private en protected

We hebben geen gebruik gemaakt van friend of globale functies/variabelen. Public hebben we wel gebruikt zodat we bijvoorbeeld functies van een speler kunnen oproepen in het bord zoals "hasJailCard", "isInJail". Voor membervariabelen hebben we nergens public gebruikt aangezien dit vaak ook ingaat tegen het principe van 'information hiding'.
Protected hebben we wel gebruikt wanneer we een abstracte klasse hadden zoals "PropertyCard" of "Space" die een aantal protected membervariabelen hadden die dan in de afgeleide klassen aangepast of opgevraagd konden worden.

## Nesting

Hier en daar twijfelden we of de nesting van if-testen of een aanroep van een functie niet te diep was. Bijvoorbeeld bij de functie 'recievePlayer->changeMoney(-((int)(card->getPriceMortgage() * 1.1)));' waarbij we 'card->getPriceMortgage() * 1.1' misschien beter eerst in een aparte variabele plaatsen en die daarna doorgeven aan de functie. Wij vonden het nog aanvaardbaar en hebben het dus zo laten staan. De '1.1' hadden we ook beter in een constante MORTGAGE_PERCENTAGE gestopt in de plaats van dit al magic number te gebruiken.
Daarnaast zijn de testen bij sommige while-lussen misschien ietwat onduidelijk, zoals 'if((card->canBuildBuildings(hasMonopoly) && (checkHasMajority(*players[currentTurn], card->getColour())) || card->getType() == PropertyCard::PropertyType::STATION))'.We hebben het in erg onduidelijke gevallen veranderd, maar in dit geval hebben we het bijvoorbeeld laten staan omdat we het duidelijk genoeg vonden.

## Destructors en memory leaks

We hebben nergens destructors toegevoegd omdat we deze allemaal samen op het einde wilden bijvoegeven wat niet zo een goede beslissing was aangezien we geen tijd meer over hadden op het einde. Hierdoor zullen er een aantal memory leaks aanwezig zijn.
Het gebruik van 'const' hebben we in sommige gevallen toegevoegd in de code. Soms hebben we dit helaas niet dadelijk gedaan waardoor we hier geen tijd meer voor hadden op het einde. We hebben het wel op de juiste manier toegevoegd in het UML-schema.

## Commentaar

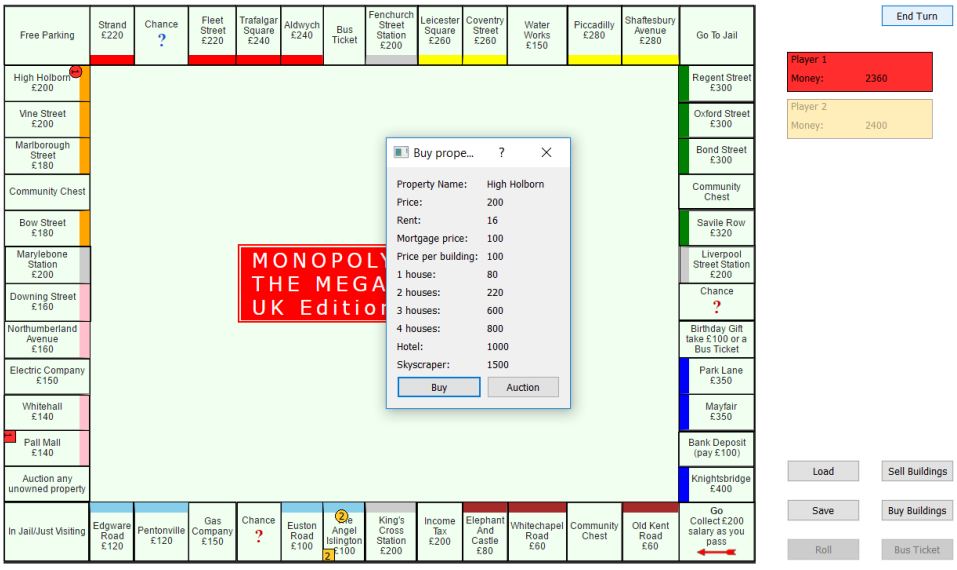
Bij de commentaar is het net hetzelfde gegaan als bij de destructors en 'const', bij de meeste complexe functies staat commentaar, maar op sommige plaatsen waar het wel nodig is, hebben we geen commentaar meer kunnen toevoegen.

## Wat we anders zouden doen

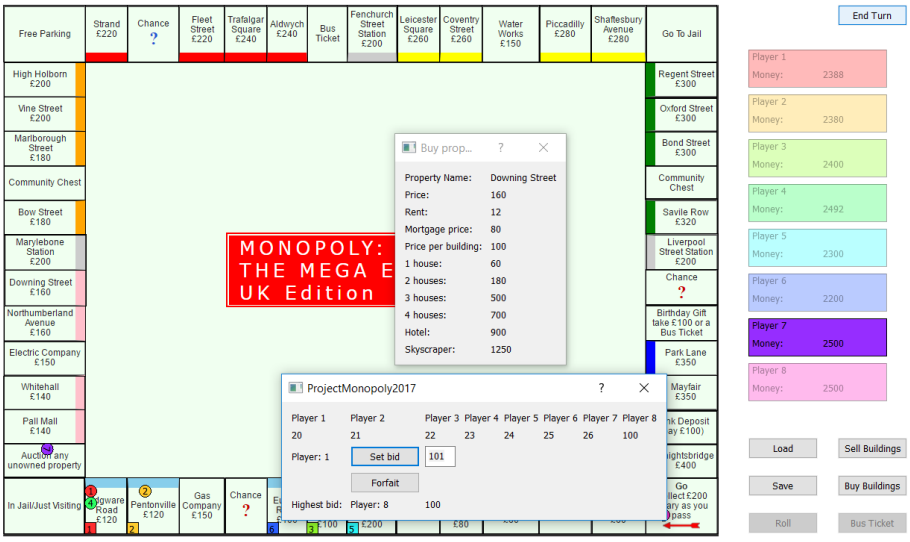
Tijdens het project zijn er toch een aantal dingen fout gelopen die we in het vervolg anders zouden doen. Bij de laatste merge hebben we gewacht tot net voor de deadline waardoor we beiden veel code geschreven hadden. In het programma dat we gebruikten om de code te mergen liep dan ook vanalles mis. Een groot deel van de code was verdwenen en deze code recuperen kostte veel kostbare tijd. We konden bepaalde delen niet meer afwerken en er was ook niet overal commentaar aanwezig.
Daarnaast is het handiger als we tijdens het schrijven van nieuwe functies dadelijk commentaar en 'const' toevoegen waar het nodig is zodat dit niet meer op het laatste moment gedaan moet worden. Bij Ho Yuet is dit vrij goed gelukt, bij Sebastiaan iets minder.
Het feit dat we geen polymorfisme gebruikten, maar dat we het type van het object checken voor het aanroepen van functies uit een bepaalde klasse, is ons vrij laat opgevallen waardoor we ook weer veel tijd verloren. We hadden misschien beter in het begin langer nagedacht over hoe we het gingen aanpakken.
Sommige magic numbers hebben ook pas vrij laat ontdekt, dit kostte gelukkig niet veel tijd, maar het was toch handig als we dit al in het begin gedaan hadden.

# Screenshots

Het kopen van een eigendom:



Het veilen van een eigendom:




Handelen van eigendommen tussen spelers: