SE2 Beroepsproduct 3 - Object georiënteerd programma

Deze toets test alle programmeervaardigheden die je tot op heden bij SE2 hebt verworven. Het is toegestaan om tijdens de toets de voor dit vak gebruikte boeken, aantekeningen en uitwerkingen (op papier of lokaal op laptop) na te slaan. Iedere vorm van communicatie is tijdens de toets verboden (bijvoorbeeld het gebruik van mail- of chatclients). Bij constatering van bovenstaande wordt de toegang tot de toets ontzegd en hier melding van gemaakt bij de examenkamer.

Duur van de toets De beschikbare tijd voor deze toets is 200 minuten. Merk op dat er geen pauzes ingeroosterd zijn: de eindtijd valt dus niet gelijk met de standaard lesuren. De toets wordt afgenomen vanaf 12:45u en eindigt om 16:05u.

Beoordeling

Bij deze toets moet je laten zien dat je de bij SE2 geleerde OOP concepten beheerst. Deze concepten komen terug in de voor dit deel van het vak opgestelde rubrics. Het uiteindelijke cijfer wordt bepaald na afloop van het verplichte feedbackgesprek. Deze feedback wordt mondeling gegeven en kan gebruikt worden voor het uiteindelijke P-assessment.

De beoordeling van de toets gebeurt op basis van de rubrics die gedefinieerd zijn voor dit beroepsproduct[1]. Je zult een voldoende behalen voor de toets als ieder leerdoel aangetoond is. Hierbij geldt vanzelfsprekend dat hoe beter en correcter het leerdoel in je oplevering verwerkt is, hoe hoger je uiteindelijke eindscore zal zijn.

Het inleveren van de toets

~~Na afronding van de implementatie pak je de map waarin je je applicatie ontwikkeld hebt in als zip- of rar-bestand. Dit bestand lever je in op Canvas bij de assignment "Toets object georiënteerd programma".~~

Voor je met je implementatie start, commit je je startproject &
dien je de URL naar je git repository.
Bij voorkeur voeg je het project als deelproject toe aan je SEBewijs.
Voeg ook Bas Michielsen toe als Member "Developer" aan je SEBewijs map.

[1] Zie voor de rubrics het blokboek van semester 2, hoofdstuk over SE2: https://portal.fhict.nl/Blokboeken/Blokboek%20S%20S2.pdf

Opdrachtomschrijving

Voor deze toets gaan we een simpel kaartspel ontwikkelen: Ezelen. Van een regulier kaartspel zijn alleen de boer, vrouw, heer en aas in het spel. Het doel van het spel is om vier kaarten van dezelfde soort (bijvoorbeeld allemaal harten) te verkrijgen. Aan het begin van een ronde krijgen alle spelers ieder vier kaarten. Iedere speler kiest een kaart uit zijn hand en geeft deze door aan de speler aan zijn linkerzijde. Dit wordt herhaald totdat iemand vier keer dezelfde soort bezit.

Een simpel voorbeeld. Bij het begin van het spel krijg je de kaarten ♠B ♠V ♠H ♥V. Aangezien je nu al drie kaarten van dezelfde soort (schoppen) hebt, geef je de harten kaart door. Van een andere speler krijg je een nieuwe kaart zodat je de volgende hand hebt: ♠B ♠V ♠H ♣A. Het ligt het meest voor de hand om vier schoppen te behalen, dus geef je de zojuist ontvangen klaveren kaart opnieuw door. Nu krijg je echter ♠A wat ervoor zorgt dat je vier kaarten van dezelfde soort hebt (♠B ♠V ♠H ♠A) en hiermee de ronde wint.

Na het aflopen van een ronde krijgen alle spelers die niet vier kaarten van dezelfde soort hebben een strafpunt. De strafpunten worden uitgedrukt in letters en vormen het woord "ezel". Na een strafpunt krijg je dus de eerste letter "e", vervolgens wordt het "ez", dan "eze" en tot slot "ezel". Wanneer een speler het gehele "ezel"-woord gevormd heeft is het spel afgelopen. De speler met het minste aantal letters is dan de winnaar.

Klassendiagram

In figuur 1 is een domeinmodel te vinden dat gemaakt is op basis van bovenstaande opdrachtomschrijving. **Merk op dat dit model nog uitgebreid zal moeten worden** omdat het op dit moment nog niet volledig en optimaal opgezet is. Het is niet nodig om deze wijzigingen te documenten middels een diagram. Onderstaande opmerkingen lichten afwegingen toe die tot het ontwerp geleid hebben.

Kaart Deze klasse beschrijft een enkele kaart. Een ♦H heeft bijvoorbeeld als soort ruiten en als waarde heer. Kaarten dienen sorteerbaar te zijn: allereerst op soort; binnen een soort wordt gesorteerd op waarde.

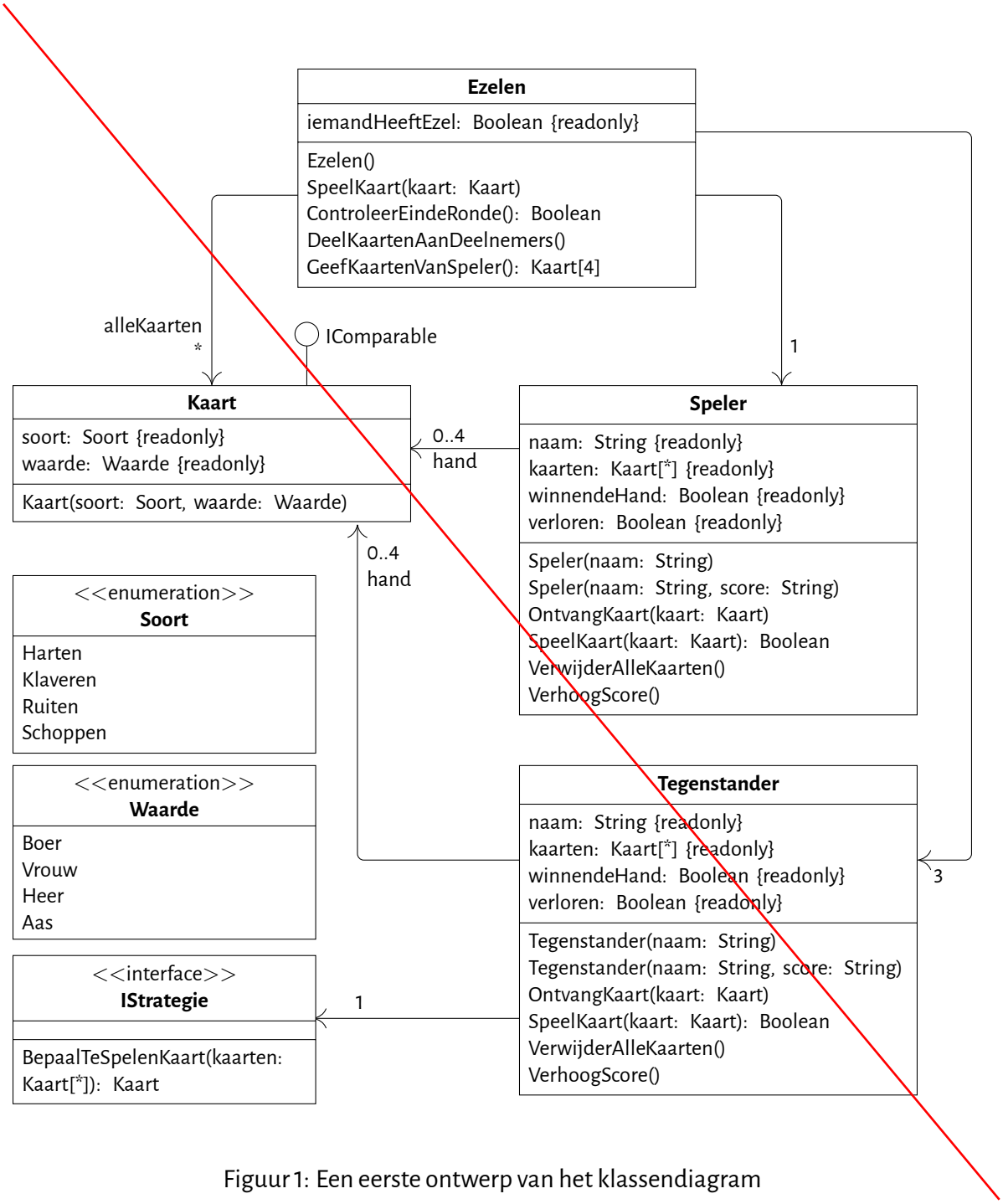
Speler Dit is de speler van het spel, oftewel de gebruiker van de applicatie. Iedere speler heeft tijdens het spelen altijd 4 kaarten in zijn hand. Met het *winnendeHand-~~attribuut~~* wordt bepaald of de speler op dit moment 4 kaarten van dezelfde soort bezit. Het *verloren-~~attribuut~~* geeft aan of de speler het spel verloren heeft, oftewel: of zijn score op dit moment "ezel" is. (Merk op dat, ook al is dit niet zichtbaar in het diagram, een speler zijn score dient bij te houden.)

methode

methode

De *OntvangKaart-* en *SpeelKaart*-operaties worden respectievelijk gebruikt om speler bij het begin van een ronde kaarten te geven en om kaarten door te geven aan andere spelers. Alleen kaarten die een speler bezit kunnen gespeeld worden. Voor het begin van een ronde dient de speler met een lege hand te beginnen (*VerwijderAlleKaarten*). Aan het einde van een ronde wordt de score van de speler verhoogd indien hij of zij geen winnende hand heeft (*VerhoogScore*).

Zie PNG



Figuur 1: Een eerste ontwerp van het klassendiagram

Tegenstander Vergelijkbaar met speler, maar de tegenstander modelleert een deelnemer die door de computer bestuurd wordt. De *SpeelKaart*-operatie zal gebruik makende van een bepaalde strategie (zie hieronder) bepalen welke kaart de tegenstander door zullen geven.

IStrategie Tegenstanders kunnen verschillende spelstrategieën hebben. Iedere strategie behoort deze interface te implementeren. Met een specifieke realisatie kan een tegenstander dan een bepaalde spelstijl hebben.

IsErEenWinnaar(): Boolean

Ezelen Dit is de kapstok-klasse van het systeem. Het formulier kan via deze klasse achterhalen of het spel afgelopen is (*iemandHeeftEzel*). De *SpeelKaart*-operatie kan gebruikt worden om vanuit het formulier aan te geven welke kaart de speler door wil geven. ~~*ControleerEindeRonde*~~ is bedoeld om te controleren of een van de deelnemers een winnende hand heeft. Is dit het geval dan wordt hierin de ronde beëindigd door punten toe te kennen aan de spelers die niet vier dezelfde kaarten hebben. Middels de return value is te bepalen of de ronde ten einde is of niet. Via *DeelKaartenAanDeelnemers* kan een nieuwe ronde begonnen worden. *GeefKaartenVanSpeler* haalt tot slot de vier kaarten op die de speler in zijn hand heeft om op het scherm te tonen.

User interface

In de user interface zijn een aantal elementen te vinden. Hieronder vind je de beoogde functionaliteit van iedere knop.

lstStand Toont alle deelnemers en de score die ze op dit moment hebben.

lstKaartenSpeler Toont de kaarten die de speler op dit moment in zijn hand heeft.

btnDoorgeven Alleen beschikbaar wanneer er een kaart in *lstKaartenSpeler* geselecteerd is. De geselecteerde kaart gaat naar tegenstander 1; alle tegenstanders geven ook hun kaarten door. Eveneens wordt hier gecontroleerd of een speler een winnende hand heeft, en wanneer dit het geval is de ronde beëindigd.

btnNieuweRonde Deze knop wordt gebruikt om een nieuwe ronde te starten. Eerst wordt gecontroleerd of een van de spelers "ezel" heeft: is dit het geval dan is het spel ten einde. Zo niet, dan worden de kaarten opnieuw geschud (zie voorbeeld aan het einde van de toets) en verdeeld onder de spelers.

Minder belangrijk dan een goede Unit test!!

btnStandOpslaan Slaat het spel op naar een door de gebruiker te bepalen locatie.

btnStandLaden Laadt het spel vanuit een door de gebruiker te bepalen locatie.

btnResetSpel Laat het spel van vooraf aan beginnen: alle scores worden naar de beginsituatie hersteld.

Requirements

Commit regelmatig, en minstens elk half uur.

Maak een applicatie die voldoet aan de gegeven opdrachtomschrijving. Om je op weg te helpen is er een project gegeven waarin al een formulier is ontworpen. Deze kun je hergebruiken, maar wijzigingen zijn natuurlijk toegestaan. Ook herhalen we hier nogmaals dat het gegeven klassendiagram niet volledig is. Om alles werkend te kunnen krijgen moet je dus mogelijk attributen, operaties en klassen toevoegen en code verplaatsen. Let op Single Responsibility en vermijden van dubbele code!

Zorg ervoor dat de opgeleverde applicatie de onderstaande functionaliteiten biedt. De genoemde punten staan in willekeurige volgorde: het is geen vereiste om in deze volgorde de implementatie te verzorgen. De geschatte tijd per punt is grotendeels gelijk: rond de 20 tot 30 minuten per onderdeel. Merk op dat de volgorde van de eisen willekeurig is; niet ieder punt is een geïsoleerde opdracht.

Commit 1

- Het klassendiagram dient correct en volledig omgezet te zijn naar code. De functionaliteit die in figuur 1 gegeven is moet terug te vinden zijn in de uiteindelijke implementatie. Veranderingen zijn toegestaan, maar alleen als deze de applicatie(structuur) ten goede komen.
- De implementatie van het klassendiagram dient verbeteringen te bevatten ten opzichte van het gegeven model. Hiervoor maak je gebruik van de object geörienteerde vaardigheden die je geleerd hebt om code te structureren, generaliseren en ontdubbelen.
- De applicatie dient op een logische manier om te gaan met fouten en incorrecte gegevens.

Commit

- Kaarten die de speler in zijn hand heeft moeten gesorteerd (eerst op soort, dan op waarde) zichtbaar zijn op het scherm.

Commit

- De huidige stand dient gesorteerd zichtbaar te zijn op het scherm: de speler met het minste aantal letters komt bovenaan.
- Bepaal per klasse of het nuttig is dat deze een *ToString*-methode bevat. Voor de speler en tegenstander moet naast de naam hierbij ook de score te vinden zijn.

minder belangrijk dan een goede unit test

- Het moet mogelijk zijn om het huidige spel op te slaan en later weer in te laden. In een tekstbestand moet voor alle deelnemers de naam en score opgeslagen worden. De huidige kaarten hoeven niet opgeslagen te worden: na het laden wordt altijd begonnen met een nieuwe ronde.

Commit

- De tegenstanders moeten via een nieuw te ontwerpen strategie spelen. Maak hiertoe een of meerdere strategieën aan gebaseerd op de gegeven interface.

Commit

- Zorg ervoor dat relevante delen van de applicatie automatisch getest worden. Bepaal zelf een aantal methodes die geschikt zijn en implementeer werkende unit tests hiervoor.

Zorg dat je minstens 1 spel helemaal speelt.

Tot slot moet het spel natuurlijk naar behoren werken en voldoen aan de spelregels zoals gegeven in de opdrachtomschrijving. Zorg er voor dat alle code op de juiste plaats staat. Probeer zo volledig mogelijk te zijn in je uitwerking en neem de moeite om extra's en uitbreidingen toe te passen als je daar reden toe ziet.

De lijst met te delen kaarten vullen

Voor het delen van de kaarten wil je alle mogelijke combinaties van soorten en waarden in een lijst hebben. Onderstaande code biedt hiervoor een mogelijke implementatie.

```
List<Kaart> kaarten = new List<Kaart>();
foreach (Soort soort in Enum.GetValues(typeof(Soort)))
{
    foreach (Waarde waarde in Enum.GetValues(typeof(Waarde)))
    {
        kaarten.Add(new Kaart(soort, waarde));
    }
}
```