MSO Lab assignment 3
Mankala, increment II

Versie 2023

Inleiding

In de vorige opgave heb je een ontwerp en basis-implementatie gemaakt voor een applicatie waarmee het Mankala-spel kan worden gespeeld. In deze opdracht breiden we het spel uit met nieuwe functionaliteit. Waar nodig pas je daarvoor je ontwerp aan. Verder zorg je voor het testen van de functies en laat je zien dat je code van goede kwaliteit kan schrijven.

Nieuwe features

Variant 'Wari'

Wari wordt meestal gespeeld met 6 speelkuiltjes per speler (maar dat kan variëren). Spelers hebben geen thuiskuiltjes. De spelers verzamelen stenen buiten de kuiltjes.

N.B. Ook al hebben spelers geen thuiskuiltjes, het speelbord kan er precies hetzelfde uitzien als het bord dat gebruikt wordt voor de mankalaregels. Het verschil is dat

verzamelkuiltjes, in tegenstelling tot de thuiskuiltjes, niet meedoen bij een zet; ze worden alleen maar gebruikt om 'gevangen' stenen in te verzamelen!

Er wordt meestal gestart met 4 stenen per kuiltje.

Na een zet, dus wanneer de stenen uit één kuiltje voor een zet zijn uitgestrooid, treedt één van de volgende situaties op:

- De laatste steen komt in een kuiltje van de tegenstander dat één of twee stenen bevatte (er liggen nu dus twee of drie stenen). De stenen worden gepakt, en bij de buit van de speler gestopt. De zet is over, de beurt is over: de tegenstander is aan de beurt.
- De laatste steen komt ergens anders terecht. De zet is over, de beurt is over: de tegenstander is aan de beurt.

Wanneer een speler geen zet kan doen doordat deze geen stenen meer heeft, eindigt het spel. De speler die de meeste stenen heeft gevangen is de winnaar.

Nog een variant

Bedenk zelf een derde variant, die regels van Mankala en Wari combineert, en misschien zelfs nog een nieuwe regel voor één van de onderdelen bevat. Geef deze variant een unieke, zelfbedachte naam.

User interface

Het programma moet het verloop van het spel visualiseren, waarbij de gebruiker steeds kan aangeven welke zet er gedaan moet worden. Je hoeft dus niet "tegen de computer" te kunnen spelen. Je mag zelf kiezen hoe je het spel wilt visualiseren, en hoe de interactie met de gebruiker verloopt. Mogelijkheden zijn bijvoorbeeld:

- Een console-programma, waarin m.b.v. Ascii-art de vakjes van het bord worden getoond, en de gebruiker via de Console het nummer van het vakje waar de nieuwe zet begint doorgeeft.
- Een Forms-programma, waarin het bord op een Panel wordt getoond, en de gebruiker het startvakje van de zet met de muis kan aanklikken.
- Eventueel een game gemaakt in Unity of MonoGame waarin het bord wordt getoond, en de gebruiker het startvakje van de zet kan selecteren. Zorg er wel voor dat het spel makkelijk kan worden gerund door de TA's!

Voordat het eigenlijke spel begint moet de gebruiker interactief kunnen kiezen welke variant gespeeld gaat worden (Mankala/Wari/eigen variant, aantal kuiltjes, aantal beginsteentjes, ...). Bij een console-programma kan dat via de command-line arguments, of via een vraag-antwoord dialoog aan het begin. Bij een grafisch georiënteerd programma kan dat bijvoorbeeld via Textboxes en Buttons op het hoofdscherm, of in een dialoog als reactie op een menukeuze. De grafische vormgeving van de UI hoeft niet fancy te zijn! Het gaat er vooral om dat er een frontend moet zijn om de game-engine te bedienen.

Opdrachten

Opdracht 1 Uitbreiding en aanpassing softwareontwerp

Breid je ontwerp uit met de bovenstaande features. Pas je ontwerp zo nodig aan zodat de nieuwe features makkelijk kunnen worden toegevoegd, en verwerk eventuele feedback van de TA's.

Beschrijf in een apart pdf-bestand de volgende onderdelen:

- **Vernieuwd ontwerp.** Werk je klassendiagram bij. Als het erg complex wordt, kun je het eventueel opsplitsen in meerdere diagrammen. Laat dan wel duidelijk zien hoe ze samenhangen. Geef van alle klassen een beschrijving van de verantwoordelijkheden, en laat zien hoe de klasse die verantwoordelijkheden uit kan voeren.

  Geef aan welke design patterns je hebt gebruikt, en geef van elk pattern aan welk deelprobleem het oplost, en hoe je het patroon gebruikt (door een vergelijking met het desbetreffende klassendiagram uit het tekstboek). Vergelijk de gekozen oplossing met alternatieven en beargumenteer waarom je voor deze oplossing hebt gekozen.

- **Afwijkingen van het originele ontwerp.** Geef een toelichting waar je bent afgeweken van het originele ontwerp in de vorige opdracht, en hoe je de feedback daarop hebt verwerkt.

- **Alternatieve user interfaces.** Leg uit hoe het ontwerp kan omgaan met de mogelijkheid van een andere user-interface (dus bijvoorbeeld een command-line interface als je Windows Forms hebt gebruikt).

Opdracht 2 Implementatie en codekwaliteit

Schrijf C#-code om de nieuwe features (varianten, UI) aan het spel toe te voegen. Zorg dat je code van goede kwaliteit is, door onder andere:

- Een code-standaard aan te houden.
- Een tool te gebruiken die feedback op je codekwaliteit geeft, zoals SonarQube of Resharper.
- Het berekenen van diverse metrieken, en het inspecteren van de resultaten daarvan.
- Te zorgen voor comments waar nodig, en duidelijke naamgeving.
- Code reviews van elkaars code te houden waarin je bovenstaande items checkt.

Beschrijf in een reflectie (ongeveer 1-2 pagina's) welke maatregelen je m.b.t. codekwaliteit hebt genomen. Geef meerdere concrete voorbeelden van refactorings[1] die je hebt doorgevoerd. Laat ook zien hoe de resultaten van toegepaste metrieken hebben geleid tot aanpassingen in de code.

[1] https://refactoring.com/catalog/

Opdracht 3 Testing

Test je applicatie uitgebreid met de volgende onderdelen:

- Unit tests. Maak waar nodig fake objecten (mocks, stubs) om code met afhankelijkheden geïsoleerd te kunnen testen.
- System tests. Gebruik user stories en scenario's om diverse functies en spelverlopen te kunnen testen. Voer deze testen regelmatig handmatig uit, en houd de resultaten (geslaagd/gefaald+reden) bij in een document (bv een Excel-sheet).

Om voor dit onderdeel 'exceptional' te scoren, heb je bovenstaande basis zeer goed uitgevoerd en aangevuld met een volgend onderdeel: property-based testing, een mocking framework, testing en evaluatie met echte users (denk aan familie, vrienden), of een andere relevante testtechniek.

Zorg voor goede kwaliteit van je geautomatiseerde tests[2], en voeg een screenshot bij van de laatste testrun. Lever ook de resultaten van de handmatige testen in. Geef een korte reflectie op de aanpak, de testkwaliteit, en de resultaten (ongeveer een halve pagina).

Opdracht 4 Taakverdeling

Geef een overzicht van de taakverdeling: wie heeft gewerkt aan welke onderdelen?

Bonusopdrachten (maximaal 1 bonuspunt)

Je kunt maximaal 1 bonuspunt verdienen, dus je kan uit maximaal twee van de onderstaande opties kiezen.

Logbestand (0.5 bonuspunt)

Implementeer met behulp van `events`[3] of het observer pattern[4] functionaliteit voor een logbestand, zodat je de gebeurtenissen van een heel spel terug kan lezen. Voeg tekst toe aan het bestand wanneer:

- De volgende speler aan de beurt is.
- De huidige speler een bepaald kuiltje kiest.
- De staat van het bord verandert (per uitgestrooide steen een log entry).

Tweede user-interface (0.5 bonuspunt)

Implementeer een alternatieve user-interface (dus bijvoorbeeld een command-line interface als je Windows Forms hebt gebruikt). Vermeld eventueel gebruikte design patterns.

[2] https://testsmells.org/

[3] https://docs.microsoft.com/en-us/dotnet/standard/events/

[4] in C# kan dit met de `IObservable<T> interface`, zie https://docs.microsoft.com/en-us/dotnet/api/system.iobservable-1?view=net-5.0

Nieuwe feature met pattern (0.5 bonuspunt)

Bedenk zelf een originele nieuwe feature waarvoor je een nog niet gebruikt design pattern kan inzetten. Licht toe in het ontwerp.

Algemene aanwijzingen

Zorg verder dat je rekening houdt met de volgende punten:

- **Taal.** De opdracht mag zowel in het Engels als Nederlands worden gemaakt.
- **Gebruik bestaande code.** De uitwerking van de opdracht is het *eigen werk* van jou en je teamgenoot. Als je gebruik maakt van code die niet van jezelf is, dien je je bron te vermelden in je verslag en in de code. Dit is overigens alleen toegestaan voor hulpmiddelen als libraries om bestanden uit te lezen, UI elementen e.d. Je mag AI tools zoals ChatGPT en Copilot *alleen* gebruiken voor het vinden van bugs in je code, en het genereren van code *binnen 1 methode*. Als je dit doet, moet je het volledige chat-transcript meesturen, en vermelden in de code welke stukken zijn gegenereerd. Algemene achtergrondinformatie over het gebruik van AI-tools in Informatica-onderwijs kun je hier vinden: https://iticse23-generative-ai.github.io/media/ITiCSE_2023_WG4_Mature_Draft_Jul_9_2023_AppendixD.pdf

  Het ontwerp van de klassen, attributen, methodes en patterns moet je eigen werk zijn. Vraag bij twijfel eerst of je iets mag gebruiken!
- **Werkverdeling.** Het werk is *evenwichtig* verdeeld, en beide teamleden leveren een bijdrage aan *alle* onderdelen: ontwerpen, programmeren, testen, en reflecteren. Als dit duidelijk niet het geval is, wordt dit meegenomen in de beoordeling.
- **Beoordeling.** Zie het bijgevoegde beoordelingsformulier (onder voorbehoud) om een indruk te krijgen van hoe je wordt beoordeeld.

Inleveren

Maak een zip- of rar-file van het complete project, inclusief files om het project te kunnen bouwen (dus de project- en solution-files als je Visual Studio gebruikt, of anders bijvoorbeeld een Makefile). Clean het project voordat je het inpakt, zodat de zipfile niet onnodig groot wordt vanwege allerlei intermediate files en executables. Maak zo nodig een readme-bestand met instructies voor het runnen van het spel.

Voeg een pdf-bestand bij met je uitwerking van de volgende onderdelen:

- Bijgewerkt ontwerp en toelichtingen (zie opdracht 1)
- Reflectie op codekwaliteit (zie opdracht 2)
- Resultaten en reflectie testing (zie opdracht 3)
- Taakverdeling (opdracht 4)