

# Inleiding Programmeren 2018-2019

## Examentaak C++

### Schaken

prof Toon Calders, Len Feremans, Tom Hofkens

Deadline: 6 januari 2018, 22u

## Belangrijke Informatie

- Maak deze opdracht *individueel*. Samenwerken is oneerlijk ten opzichte van je medestudenten. Je wordt gecontroleerd en bij samenwerking zijn we zeer streng. Bij het gebruik van code op het internet, vermeld je de bron. Het spreekt voor zich dat een kort fragment kopiëren uit een tutorial toegelaten is, maar een volledige oplossing specifiek aan de opdracht niet. Er wordt gebruik gemaakt van software voor plagiaat controle waarbij ook wordt vergeleken met online beschikbare code fragmenten.
- Tijdens de examenperiode zullen alle studenten die de examentaak indienden mondeling ondervraagd worden over hun code. Het doel van het mondelinge gedeelte is na gaan of jij zelf de auteur bent van de code. Je hoeft je hier niet specifiek op voor te bereiden; kort de dag voordien jouw code even herbekijken volstaat.
- Bij de beoordeling houden we ook rekening met de kwaliteit van de code. Let er op dat je niet onnodig code dupliceert; dat de complexiteit van elke functie beperkt is (bijvoorbeeld door hulp functies te definiëren); dat je variabelen en functies duidelijke namen geeft; dat je je houdt aan code conventies; dat je complexe code goed documenteert.
- Benodigdheden: C++ compiler, we werken met de C++ 11 standaard.
- De deadline is op *zondag 6 januari om 22:00*.
- Dien de opdracht in op Blackboard, onder *opdrachten*. Je uploadt een *zip* bestand met alle .cpp en .h bestanden.

## Omschrijving

In deze opdracht maak je een schaakspel dat door twee personen gespeeld kan worden. Het schaakspel laat enkel geldige zetten toe en test op schaakmat. De opdracht is opgedeeld in onderdelen van stijgende complexiteit. Afhankelijk van welke onderdelen correct geïmplementeerd werden, wordt het uiteindelijke resultaat voor deze opdracht bepaald. Het is dus niet nodig om alle onderdelen te implementeren om te slagen voor deze opdracht. Naast de beoordeling van de verschillende onderdelen van de opdracht, wordt ook de programmeerstijl beoordeeld. Hiermee wordt het schrijven van commentaren, const correctness, modulaire programmeren, het gebruiken van de juiste data- en controlestructuren en het schrijven van overzichtelijke code bedoeld. Een uitzonderlijk goede of uitzonderlijk zwakke programmeerstijl kan het eindresultaat met 20% beïnvloeden.

## Opdracht

De regels van het schaakspel vind je op:

<http://www.schaakzone.nl/schaken-voor-beginners/spelregels-van-schaken.php>.

Negeer voorlopig de bijzondere spelregels “en passant”, “rokade” en “promotie”.

Mocht je niet vertrouwd zijn met schaken, aarzel dan niet om een van de assistenten of de docent aan te spreken voor een korte introductie; deze opdracht heeft als doel jouw programmeervaardigheden te testen, niet jouw kennis van het schaakspel!

Voor dit project wordt reeds code beschikbaar gesteld. Deze code omvat de volgende bestanden:

- *chessboard.cpp* en *chessboard.h*: Deze bestanden bevatten de code om het schaakbord te tekenen. **Pas deze bestanden NIET aan.**
- *main.cpp*: Dit bestand opent een venster dat het schaakbord bevat. In principe hoeft je ook in dit bestand niets te wijzigen.
- *resources.h*: Dit bestand bevat een string `path` die je kan aanpassen indien de figuren niet correct gevonden worden (als er geen schaakstukken worden getoond). In principe moet je ook dit bestand niet aanpassen.
- directory *resources* met .svg bestanden: dit zijn de tekeningen van de schaakstukken. *chessboard.cpp* laadt de figuren uit deze bestanden. Als deze figuren niet worden weergegeven, dan moet je waarschijnlijk de variabele `path` in *resources.h* aanpassen.
- *mainwindow.cpp* en *mainwindow.h*: in deze bestanden zal je een (klein) deel van jouw code moeten schrijven. Er staat hier wat voorbeeld code waaraan je kan zien hoe je het grafische schaakbord moet gebruiken. Een vrijblijvende tip: probeer het aantal lijnen code die je hier schrijft te beperken; implementeer de logica van het schaakspel zoveel mogelijk in de bestanden die beschreven worden in volgend punt.
- *game.cpp*, *game.h*, *schaakstuk.cpp*, *schaakstuk.h*: deze bestanden gaan de kern vormen van jouw project. Jij hebt de volledige vrijheid over hoe je deze bestanden aanpast. Voeg const, virtual, constructors, member functies en variabelen, access specifiers (public/private/protected) ... toe waar je dat wenselijk of nodig acht.
- *positie.cpp*, *positie.h*: Deze bestanden geven de implementatie van een klasse positie die je kan gebruiken om de positie van schaakstukken te beschrijven.
- Je mag ook nieuwe bestanden toevoegen aan jouw project.

## 1. The essentials (50%)

Uit deze sectie moet je verplicht alle functionaliteit implementeren

### 1.2 Zet het bord al maar klaar ... (10%)

Breid de klasse `game` in bestand *game.cpp* uit als volgt:

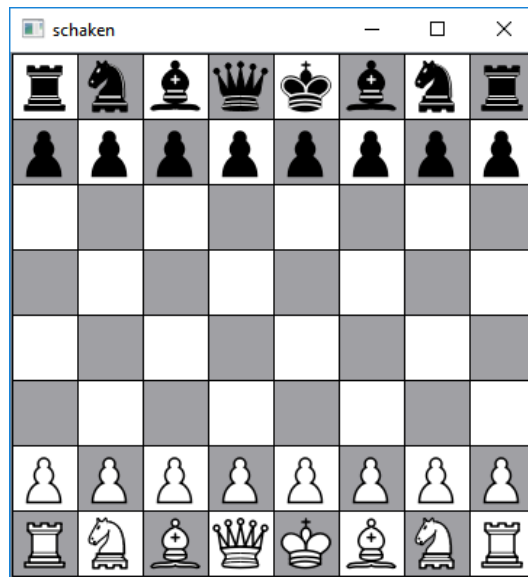
1. Voeg member variabelen toe om een speelbord met schaakstukken te representeren. **Denk goed na over jouw keuze van data structuur om schaakstukken in te bewaren. Een goede keuze kan je veel werk besparen ...**
2. Voeg member functies toe om schaakstukken op het bord te plaatsen, en om op te vragen welk schaakstuk op een bepaalde positie op het speelbord staat.

- Voeg een member functie `setStartBord()` toe aan klasse `game` die het bord klaar maakt voor een nieuw spel (merk op dat hier nog geen grafische weergave gevraagd wordt!)

### 1.3 en maak het zichtbaar (10%)

Voeg aan de klasse `mainwindow` in bestanden `mainwindow.h` en `mainwindow.cpp`, de definitie van de functie `update()` toe die de huidige staat van het spel grafisch weergeeft. Hiervoor zal je de posities van de stukken uit `game g` moeten halen en de functie `scene->setItem(int,int,Piece)` moeten gebruiken. Het bestand `mainwindow.cpp` bevat voorbeeldcode die het gebruik van de scene variabele illustreert.

Pas nu het programma zodanig aan dat bij de start van het programma de start opstelling getoond wordt. Als alles goed ging krijg je nu bij het opstarten van het spel volgend venster te zien:



### 1.4 A game of Pawns (15%)

Nu het bord klaar staat, is het tijd om te beginnen spelen. We starten met de bewegingen van de pion. Je hoeft in deze fase geen rekening te houden met “schaak” en “schaakmat”.

- Voeg een methode `vector<positie> geldige_zetten(game&)` toe aan de klassen `Schaakstuk` en `Pion`. Deze methode geeft een vector met alle posities waarnaar een schaakstuk zich kan verplaatsen op het gegeven speelbord. Om dit onderdeel te halen, is het voldoende indien voor de andere schaakstukken deze methode een lege lijst geeft.

```
// Dit is "pseudo code"; jouw code kan er mogelijk licht anders uit zien
game g;
g.setStartBord();
schaakStuk* s=g.getSchaakstuk(positie(1,0)); // Zwarte pion
vector<positie> v=s->geldige_zetten();          // v bevat de posities (2,0) en (3,0)
```

- Voeg een methode `bool move(schaakStuk*,positie);` toe aan de klasse `game`. Deze methode verplaatst het schaakstuk naar de nieuwe positie indien dit toegestaan is en geeft dan `true` terug. Indien de zet niet is toegestaan, wordt het stuk niet verplaatst, en wordt `false` teruggegeven.

```
g.move(s,positie(2,0)); // Geeft true; het stuk wordt verplaatst
g.move(s,positie(5,1)); // Geeft false; het stuk wordt niet verplaatst
```

## 1.5 Iedereen speelt mee (20%)

Breid de functie `geldige_zetten(game&)` uit naar de andere schaakstukken. Je moet de methode `geldige_zetten(game&)` dus implementeren voor de volgende klassen:

- Toren
- Paard
- Loper
- Koning
- Koningin

Pas indien nodig de methode `bool move(schaakStuk*,positie);` aan in de klasse `game` om de bewegingen van de andere stukken mogelijk te maken. Opnieuw hoef je geen rekening te houden met “Schaak” en “Schaakmat”.

## 1.6 Move it! (5%)

Pas de functie `void MainWindow::clicked(int r, int k)` aan zodat je via de interface jouw zetten kan uitvoeren. De eerste keer dat je klikt selecteer je het stuk dat je wil bewegen. De tweede keer dat je klikt bepaal je de positie waarnaar het stuk verplaatst moet worden. Ga na of het een geldige zet is, en indien dit zo is, doe de verplaatsing in het spel (`game g`) en update de grafische weergave. *Houd er rekening mee dat wit begint en de spelers wit en zwart daarna afwisselend spelen.*

## 2. Optioneel deel: opslaan en laden van een spel (10%)

Bekijk de functies `open` en `save` in `MainWindow.cpp`. Werk deze functies verder af zodat een spel bewaard en geladen kan worden. Implementeer eveneens `new` om een nieuw spel te starten. Deze functies worden automatisch uitgevoerd wanneer een gebruiker ze in het file-menu selecteert.

## 3. Optioneel deel: Undo en Redo (20%)

Soms kunnen we ons al wel eens vergissen van zet. Voeg undo en redo operaties toe, door de functies `undo` en `redo` in `MainWindow.cpp` te voltooien. Je zal ook andere delen van jouw code moeten aanpassen om deze functionaliteit toe te voegen.

## 4. Optioneel deel: schaak en schaakmat (30%)

Een essentieel onderdeel van het schaakspel, je raadt het nooit, zijn de begrippen “schaak” en “schaakmat”.

### 4.1 Check ... (10%)

- Voeg aan de klasse `game` een methode `schaak(zw)` toe, die aangeeft of de koning van de gegeven kleur al dan niet schaak staat.
- Pas de methode `move(schaakStuk*,positie)` van de klasse `game` of method `geldige_zetten(game&)` van de schaakstukken aan zodat zetten die ervoor zorgen dat de eigen koning schaak komt te staan, niet toegestaan zijn.

## 4.2 ... and mate (15%)

- Implementeer een methode `schaakmat(kleur)` die aangeeft of de koning van de aangegeven kleur al dan niet schaakmat staat.
- Implementeer een methode `pat(kleur)` die aangeeft of de koning van de aangegeven kleur al dan niet pat staat. Pat wil zeggen dat de koning niet schaak staat, maar dat er geen enkele geldige beweging van eender welk stuk van die kleur mogelijk is.

## 4.3 Ik wilde je even laten weten ... (5%)

Geef een boodschap als een speler schaak, schaakmat, of pat staat. Bij schaakmat en pat stopt het spel; dit wil zeggen: er zijn geen verdere zetten meer nodig.



## 5. Optioneel deel: Voor de volledigheid ... (20%)

Voeg de bijzondere spelregels “en passant”, “rokade” en “promotie” toe.

## 6. Optioneel deel: Help! (10%)

Voeg visuele elementen toe die de gebruiker helpen bij het bepalen van zijn of haar zetten. Bijvoorbeeld:

- (5%) Als je op een stuk klikt om het te verplaatsen, markeer dan dit stuk (gebruik `scene->setPieceFocus(...)`).
- (5%) Als je op een stuk klikt om het te verplaatsen, geef dan alle geldige posities waarnaar dit stuk verplaatst kan worden (gebruik `scene->setTileFocus(...)`).



## 7. Optioneel deel: AI (20%)

Maak een computer speler. De computer speler selecteert steeds random een schaakstuk en beweegt dit stuk naar een random positie. Echter, als de computer speler een stuk van de tegenstander kan schaakmat of schaak kan zetten, of een stuk kan slaan, zal deze zet de voorkeur krijgen op andere, meer willekeurige zetten.