Inleiding Programmeren

Labo-examen C++

1B Informatica,
Academiejaar 2024-2025

Tom Hofkens
tom.hofkens@uantwerpen.be

Tim Apers
tim.apers@uantwerpen.be

15 januari 2025

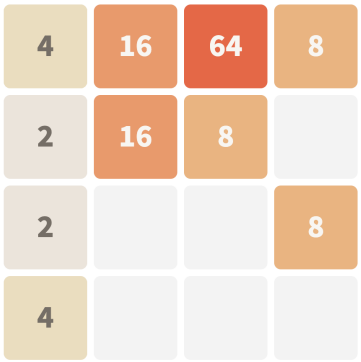
Belangrijke informatie

- Maak deze opdracht individueel. Het gebruik van communicatiemiddelen zoals smartphones en smartwatches is niet toegestaan. Internet mag enkel gebruikt worden om informatie zoals C++ documentatie op te zoeken. Kopiëren van online beschikbare code is strikt niet toegestaan. Dit omvat ook het gebruik van tools die op basis van een beschrijving code genereren, bestaande code uitbreiden, of commentaren toevoegen.
- Je hebt **2,5u** tijd om het examen op PC te maken.
- Voorzie elk bestand dat je aanmaakt of aanpast van een hoofding die jouw naam en studentnummer bevat.
- Je mag gebruik maken van de documentatie die via Inginious beschikbaar is.
- Dien je praktijk-oplossing in via INGInious.
- Er zijn 7 vragen. De meeste vragen zijn onafhankelijk van elkaar op te lossen. Het totaal aantal punten telt op tot 22 en met programmeerstijl kan jouw resultaat tot 2 punten beïnvloeden in zowel positieve als negatieve zin. De som van de punten op alle delen die je correct oploste is jouw definitieve punt, tenzij dat hoger is dan 20. In dat geval behaal je 20/20.

Inleiding

Lees zorgvuldig onderstaande beschrijving voordat je aan het examen begint. Een goed begrip van de code waarvan je start is essentieel!

We gaan in deze opdracht het spelletje "2048" implementeren. 2048 is een puzzelspel dat gespeeld wordt op een veld met 4 rijen en 4 kolommen. Elke cel van het veld kan ofwel leeg zijn, ofwel een getal bevatten dat een macht van 2 is (bijvoorbeeld 2, 4, 8, enz.). Het doel van het spel is om door het samenvoegen van gelijke getallen een tegel met de waarde 2048 te verkrijgen. Een mogelijke opstelling van het spelbord kan er bijvoorbeeld als volgt uitzien:



Je speelt 2048 door telkens het bord in één van de vier richtingen te schuiven: omhoog, omlaag, naar links, of naar rechts. Bij elke schuifbeweging worden de volgende regels gevolgd:

1. Alle tegels schuiven in de gekozen richting zo ver mogelijk op, totdat ze een andere tegel of de rand van het bord tegenkomen.
2. Wanneer twee tegels met dezelfde waarde tegen elkaar schuiven, worden ze samengevoegd tot één tegel waarvan de waarde de som van de oorspronkelijke tegels is.
3. Bij elke schuifbeweging verschijnt er een nieuwe tegel met waarde 2 op een willekeurige lege plaats op het bord.
4. Het spel eindigt wanneer er geen geldige schuifbewegingen meer mogelijk zijn.

Door deze regels te gebruiken, kunnen we strategisch proberen het bord te manipuleren. Bijvoorbeeld: als er in een rij twee naast elkaar liggende tegels met waarde 4 staan, kunnen we door in de juiste richting te schuiven een tegel met waarde 8 creëren. Op die manier kunnen we het spel verder spelen.

Bestanden

Bekijk de gegeven code. Die bestaat uit 11 bestanden:

- `_main.cpp`: het hoofdprogramma
- Telkens 2 bestanden met code voor de klasses:
  - Game.
  - Tile en haar direkte subklasses WallTile, ValueTile, en EmptyTile;

Om van start te gaan, doorloop je volgende stappen. Doe het exact zoals hier staat, anders ga je problemen krijgen met de antivirus software die zal zorgen dat je bestanden in quarantaine komen te staan waardoor compileren en uitvoeren enorm traag worden.

1. Maak een nieuw CLionproject aan IN DE MAP CLionProjects. Dus niet in Downloads of Documenten of ...
2. Verwijder de main.cpp die automatisch werd aangemaakt.
3. Kopieer alle bestanden van de map startcode naar je project.
4. Open CMakeLists.txt.
5. Voeg bij `add_executable`, na de naam van je project volgende lijstje toe:

   ```
   _main.cpp Game.h Game.cpp Tile.h Tile.cpp ValueTile.h ValueTile.cpp
   EmptyTile.h EmptyTile.cpp WallTile.h WallTile.cpp
   ```

   Als je project 'examen' heet, dan staat er nu `add_executable(examen _main.cpp Event.cpp ...)`. Je verwijdert eventueel de main.cpp die er al stond. Let op: onze main heet `_main.cpp` (met een underscore). Op die manier staat dat bestand bovenaan en kan je makkelijk alles zippen zonder de main.
6. Links in de project browser (waar al je bestanden staan in CLion), rechterklik je op Cmakelists.txt en vervolgens op 'reload CMake project'.
7. Run de code. Die zou moeten compileren zonder fouten.

Opdracht (op 20)

Vraag 1: Code uitbreiden (3pt)

Voer de volgende opdrachten uit om de verschillende soorten tegels te implementeren. Elke klasse moet een `visualize` methode bevatten die een karakter teruggeeft ter representatie van de tegel op het bord. Deze methode wordt overgeërfd van de abstracte super klasse `Tile`.

1. Breid de klasse `WallTile` uit en implementeer de methode `visualize`. Het karakter van een `WallTile` komt overeen met een 'W'.
2. Doe hetzelfde voor een klasse `EmptyTile`. Het karakter is hier een underscore '_'.
3. Tot slot, implementeer een klasse `ValueTile` die een macht van 2 kan bevatten. Maak voor deze klasse ook een gepaste constructor. Aangezien de `visualize` methode een karakter moet teruggeven en sommige machten van 2 groter zijn dan 8, gebruiken we hier een verkorte karaktermapping om de complexiteit te beperken. Dit wil zeggen dat we ons voor deze opdracht beperken tot 128 en niet tot 2048.

Waarde	Karakter
2	'2'
4	'4'
8	'8'
16	'1'
32	'3'
64	'6'
128	'9'

Voorbeeld:

Volgend stukje code geeft onderstaande output:

```
1  WallTile* wall = new WallTile();
2  EmptyTile* empty = new EmptyTile();
3
4  cout << wall->visualize() << endl;
5  cout << empty->visualize() << endl;
6
7  for (int i = 2; i <= 128; i *= 2) {
8    ValueTile* value = new ValueTile(i);
9    cout << value->visualize() << endl;
10   delete value;
11 }
```

```
W
_
2
4
8
1
3
6
9
```

Vraag 2: Initialiseer het spelbord (2pt)

Implementeer de constructor voor de klasse `Game`. De constructor moet het spelbord dynamisch initialiseren op basis van een opgegeven `size`, zoals weergegeven in het onderstaande voorbeeld. Het spelbord bestaat uit een vierkant raster waarbij:

- De randen (W) worden voorgesteld door `WallTile` objecten.
- De binnenkant (_) wordt voorgesteld door `EmptyTile` objecten.

Gebruik hierbij de private membervariabele `map<pair<int, int>, Tile*> tiles;` waarbij de `pair<int, int>` de coördinaten op het spelbord voorstelt en de `Tile*` een pointer is naar een tegelobject. Het element op positie 0,0 vind je dus met deze code. Let op dat het coördinaten zijn en geen rij en kolom paren! De x-as loopt van links naar rechts, de y-as van boven naar onder zoals in de meeste games.

```
eerste = tiles[{0,0}];
onderDeEerste = tiles[{0,1}];
naastDeEerstse = tiles[{1,0}];
```

Voorbeeld:

Volgend stukje code moet onderstaande output genereren:

```
Game* game = new Game(4);
game->render();
```

```
WWWWWW
W____W
W____W
W____W
W____W
WWWWWW
```

Vraag 3: Waarden instellen (2pt)

Implementeer de methode `setValue` in de klasse `Game`. Deze methode maakt een nieuwe `ValueTile` aan met de waarde `value` op de positie (x, y). Zorg ervoor dat als deze functie een andere tile (e.g. `EmptyTile`) overschrijft, dat deze geen memory leaks veroorzaakt!

Je hoeft voor deze vraag ook niet na te gaan of een gegeven positie (x, y) geldig is.

Voorbeeld:

Volgend stukje code moet onderstaande output genereren:

```
Game* game = new Game(4);
game->setValue(1, 1, 4);
game->setValue(1, 2, 2);
game->setValue(1, 3, 2);
game->setValue(1, 4, 4);
game->setValue(2, 1, 16);
game->setValue(2, 2, 16);
game->setValue(3, 1, 64);
game->setValue(3, 2, 8);
game->setValue(4, 1, 8);
game->setValue(4, 3, 8);
game->render();
```

```
WWWWWW
W4168W
W218_W
W2__8W
W4___W
WWWWWW
```

Vraag 4: Implementeer de handleReplaceBy-methode (3pt)

Implementeer de methode `handleReplaceBy` voor elke specifieke tegelklasse (`WallTile`, `EmptyTile`, `ValueTile`). Deze methode bepaalt of een tegel vervangen kan worden door een andere tegel en geeft een bool-waarde terug. De regels zijn als volgt:

1. een `WallTile` kan nooit vervangen worden door een andere tegel en geeft altijd `false` terug.
2. Een `EmptyTile` kan alleen vervangen worden door een ValueTile. In dat geval geeft de methode `true` terug. Al de rest geeft `false` terug.
3. Een `ValueTile` kan alleen vervangen worden door een andere `ValueTile` als beide dezelfde waarde hebben. Indien de waarde overeenkomt, geeft de methode `true` terug, anders `false`.

Opgepast, de logica van de `handleReplaceBy` functie is omgekeerd tegen over de `replaceBy` functie. Dit betekent dat als je het gedrag wilt implementeren van het vervangen van een `EmptyTile` met een `ValueTile`, dat je de functie `ValueTile::handleReplaceBy(EmptyTile* tile)` moet aanvullen.

Voorbeeld:

Volgend stukje code moet onderstaande output genereren:

```
1  ValueTile* value1 = new ValueTile(2);
2  ValueTile* value2 = new ValueTile(2);
3  ValueTile* value3 = new ValueTile(4);
4  WallTile* wall = new WallTile();
5  EmptyTile* empty = new EmptyTile();
6
7  cout << value1->replaceBy(value2) << endl;
8  cout << value1->replaceBy(value3) << endl;
9  cout << value1->replaceBy(wall) << endl;
10 cout << value1->replaceBy(empty) << endl;
11 cout << wall->replaceBy(value1) << endl;
12 cout << wall->replaceBy(wall) << endl;
13 cout << wall->replaceBy(empty) << endl;
14 cout << empty->replaceBy(value1) << endl;
15 cout << empty->replaceBy(wall) << endl;
16 cout << empty->replaceBy(empty) << endl;
```

```
1
0
0
0
0
0
0
1
0
0
```

Vraag 5: Quantize the direction (4pt)

Implementeer de methode `quantizeDirection` in de klasse `Game`. Deze methode moet de richting die de speler opgeeft, kwantiseren naar één van de vier hoofdrichtingen:

- (0, -1) betekent een beweging omhoog
- (-1, 0) betekent een beweging naar links
- (1, 0) betekent een beweging naar rechts
- (0, 1) betekent een beweging omlaag

De richting wordt bepaald door de dominante component van het `direction`-paar:

- Als de absolute waarde van de x-component groter is dan die van de y-component, dan moet de beweging **horizontaal** zijn:
  - Positieve x-component $\rightarrow$ beweging naar rechts $(1, 0)$
  - Negatieve x-component $\rightarrow$ beweging naar links $(-1, 0)$
- Als de absolute waarde van de y-component groter is dan die van de x-component, dan moet de beweging **verticaal** zijn:
  - Positieve y-component $\rightarrow$ beweging omlaag $(0, 1)$
  - Negatieve y-component $\rightarrow$ beweging omhoog $(0, -1)$

Voorbeeld:

Volgend stukje code moet onderstaande output genereren:

```
Game* game = new Game(4);
pair<int, int> result1 = game->quantizeDirection({3, 1});
pair<int, int> result2 = game->quantizeDirection({1, 5});
pair<int, int> result3 = game->quantizeDirection({-4, 2});
pair<int, int> result4 = game->quantizeDirection({0, -7});

cout << result1.first << "," << result1.second << endl;
cout << result2.first << "," << result2.second << endl;
cout << result3.first << "," << result3.second << endl;
cout << result4.first << "," << result4.second << endl;
```

```
1, 0
0, 1
-1, 0
0, -1
```

Vraag 6: Beweeg een tile (4pt)

Implementeer de methode `moveTile` in de klasse `Game`. Deze methode probeert de tegel op positie `pos` te verplaatsen in de opgegeven richting `direction`. Maak bij het implementeren van deze functie gebruik van de eerder aangevulde `replaceBy` functie. De methode geeft een bool terug:

- `true` als de tegel succesvol verplaatst werd.
- `false` als de tegel niet verplaatst kon worden.
- Als de volgende coördinaat niet bestaat (i.e. zich niet bevindt in de map `tiles`), dan geeft de functie altijd `false` terug.

en gaat correct om met de volgende situaties:

- Lege tegels (`EmptyTile`) worden vervangen door een `ValueTile`.
- `ValueTile`-objecten kunnen samensmelten als ze dezelfde waarde hebben, waarbij de waarde van de resulterende tegel verdubbeld wordt (bijvoorbeeld $2 \rightarrow 4$, $4 \rightarrow 8$). TIP: Om de value van een `ValueTile` te verdubbelen raden we aan om deze aan te passen in de `handleReplaceBy` functie.

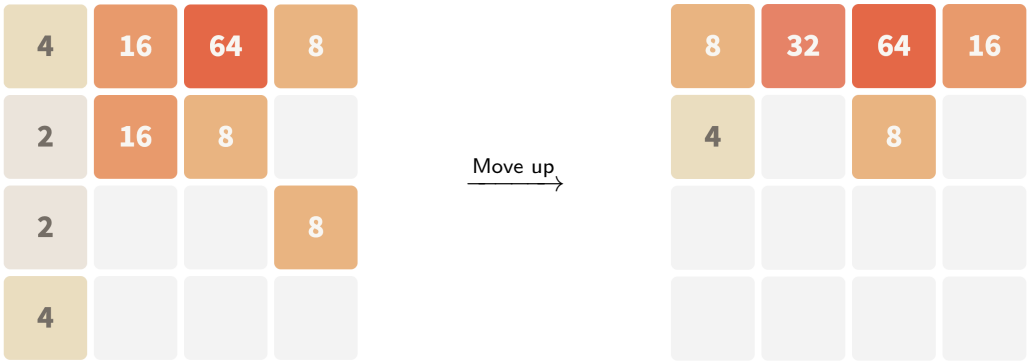
Voorbeeld:

```
Game* game = new Game(4);
game->setValue(1, 1, 4);
game->setValue(1, 2, 2);
game->setValue(1, 3, 2);
game->setValue(1, 4, 4);
game->setValue(2, 1, 16);
game->setValue(2, 2, 16);
game->setValue(3, 1, 64);
game->setValue(3, 2, 8);
game->setValue(4, 1, 8);
game->setValue(4, 3, 8);
cout << game->moveTile({2, 2}, {0, -1}) << endl;
cout << game->moveTile({4, 3}, {1, 0}) << endl;
cout << game->moveTile({4, 3}, {0, -1}) << endl;
cout << game->moveTile({4, 2}, {0, -1}) << endl;
cout << game->moveTile({5, 4}, {-1, 0}) << endl;
cout << game->moveTile({3, 2}, {0, -1}) << endl;
cout << game->moveTile({1, 4}, {1, 0}) << endl;
game->render();
```

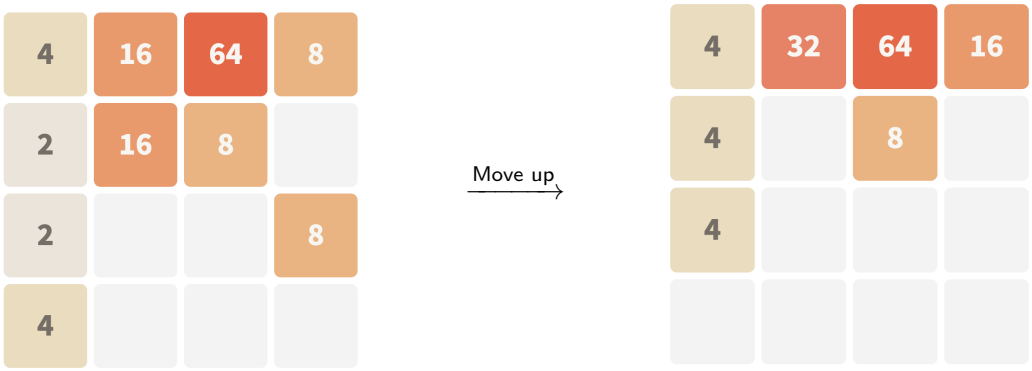
```
1
0
1
1
0
0
1
WWWWWW
W4361W
W2_8_W
W2___W
W_4__W
WWWWWW
```

Vraag 7: Het spel finaliseren - DEBUG (2pt)

De functie `move` werd al voor jullie geïmplementeerd. Deze methode beweegt het volledige spelbord in een opgegeven richting (`direction`). Momenteel is er echter een probleem met deze implementatie. De functie `move` blijft het spelbord herhaaldelijk in de gekozen richting verplaatsen zolang er nog tegels kunnen bewegen. Hierdoor ontstaat een ongewenste situatie waarin bepaalde tegels meer dan één keer bewegen binnen dezelfde actie. Dit betekent dat zich de volgende situatie voordoet:



Zoals je kan zien verplaatst bij het omhoog bewegen de meest linkse kolom zich twee keer. Echter, de tegels zouden slechts één keer mogen bewegen, en samensmeltingen mogen enkel gebeuren als de tegels dezelfde waarde hebben.



Los dit probleem op door de huidige implementatie van de methode `move` aan te passen. Zorg ervoor dat samensmeltingen van een `ValueTile`, maar één keer gebeuren bij elke oproep van `move`. Je mag hiervoor alle functies in de codebase aanpassen, inclusief de door ons voorziene `move`-functie.

Voorbeeld:

Volgend stukje code moet onderstaande output genereren:

```
1  Game* game = new Game(4);
2  game->setValue(1, 1, 4);
3  game->setValue(1, 2, 2);
4  game->setValue(1, 3, 2);
5  game->setValue(1, 4, 4);
6  game->setValue(2, 1, 16);
7  game->setValue(2, 2, 16);
8  game->setValue(3, 1, 64);
9  game->setValue(3, 2, 8);
10 game->setValue(4, 1, 8);
```

```
11 game->setValue(4, 3, 8);
12
13 game->render();
14 game->move({0, -1}); // Up
15 game->render();
16 game->move({1, 0}); // Right
17 game->render();
18 game->move({0, 1}); // Down
19 game->render();
20 game->move({-1, 0}); // Left
21 game->render();
22 game->move({1, 0}); // Right
23 game->render();
24 game->move({0, 1}); // Down
25 game->move({0, 1}); // Down
26 game->render();
27 game->move({-1, 0}); // Left
28 game->move({1, 0}); // Right
29 game->move({0, -1}); // Up
30 game->render();
```

```
WWWWWW
W4168W
W218_W
W2__8W
W4___W
WWWWWW
WWWWWW
W4361W
W4_8_W
W4___W
W____W
WWWWWW
WWWWWW
W4361W
W__48W
W___4W
W____W
WWWWWW
WWWWWW
W____W
W___1W
W__68W
W4344W
WWWWWW
WWWWWW
W____W
W1___W
W68__W
W438_W
WWWWWW
WWWWWW
W____W
W___1W
```

```
W__68W
W_438W
WWWWWW
WWWWWW
W____W
W____W
W__6_W
W_433W
WWWWWW
WWWWWW
W__49W
W____W
W____W
W____W
WWWWWW
```

Programmeerstijl (+/-2pt)

Alle methodes zijn al gegeven, maar zijn niet noodzakelijk const correct. Dat is jouw taak. Enkel de render methode mag je niet aanpassen (zie volgend jaar waarom).

Naast de opdrachten staan er ook 2 punten op programmeerstijl; dit houdt onder andere in: correct gebruik van de stl data types, const correctness, duidelijke, leesbare en goed gedocumenteerde code, gestructureerd werken zoals het doordacht gebruik maken van methodes/functies waar nodig en in mindere mate efficiëntie van de code.