# Documentatie Grafische Bibliotheek

Bjarno Oeyen
Carlos Rojas Castillo
(programmeerproject1@soft.vub.ac.be)

## 1 Inleiding

Dit document bevat een beschrijving van de operaties die voorzien worden in de grafische bibliotheek. Het doel van de grafische bibliotheek is om het mogelijk te maken om vanuit een R5RS-programma een window aan te maken en door middel van lagen en tiles er op te tekenen. Deze bibliotheek legt alle operaties van de grafische bibliotheek uit en licht toe hoe de grafische bibliotheek gebruikt kan worden voor een spellus te maken.

### 1.1 Inladen van de Grafische Bibliotheek

De grafische bibliotheek is alleen beschikbaar via de cursusruimte van PROGRAMMEERPROJECT 1 op CANVAS. Download de laatste versie van de bibliotheek, en plaats deze in dezelfde map als je eigen project. Vervolgens kan je deze inladen in je Scheme-programma. Indien je gebruik maakt van R5RS kan je door onderstaande lijn code toe te voegen, de functionaliteit van de grafische bibliotheek beschikbaar maken.

```
1 (#%require "Graphics.rkt")
```

Figuur 1: De grafische bibliotheek importeren in een R5RS-programma.

De `#%require` special form is een toevoeging van de implementatie van R5RS die aanwezig is in (Dr)Racket, en is dus zelf niet gedocumenteerd in de R5RS-specificatie. De exacte werking van deze special form is niet belangrijk voor het project, maar je kan deze special form ook gebruiken om procedures van Racket te importeren die nuttig kunnen zijn voor je project (zoals `error` voor het tonen van een foutmelding in de Read-Eval-Print Loop en `random` voor het genereren van getallen). Aangezien deze geen deel uitmaken van de grafische bibliotheek zal er in dit document niet meer uitleg over deze functionaliteit gegeven worden. Meer informatie hierover kan echter wel teruggevonden worden in het testbestand van de grafische bibliotheek (zie Sectie 1.2).

Indien je gebruik maakt van een andere variant van Scheme (bijvoorbeeld R6RS, R7RS of Racket) dan zal je op een andere manier deze grafische bibliotheek moeten importeren. Merk op dat de grafische bibliotheek zelf geschreven is in Racket, en dus alleen gebruikt kan worden in Scheme-implementaties die werken via (Dr)Racket.

### 1.2 Testbestand

Je kan op CANVAS ook een testbestand vinden waarmee je snel de belangrijkste functionaliteiten van de grafische bibliotheek mee kan testen. Plaats dit bestand in dezelfde map als de grafische bibliotheek zelf, en lees de instructies bovenaan het testbestand om te bepalen hoe je het uitvoert. We raden je aan om het testbestand samen met deze documentatie door te nemen, om te leren hoe de grafische bibliotheek werkt.

# 2 Overzicht

De procedures weergegeven in Tabel 1 zijn beschikbaar nadat de grafische bibliotheek is ingeladen (zie Sectie 1.1). Deze top-level procedures zijn constructor-procedures die bepaalde grafische elementen, geabstraheerd als ADTs, aanmaken.

| **Naam** | **Argumenten** | **Signatuur** |
|---|---|---|
| `make-window` | width, height, title, [max_fps][1] | number, number, string, [number] → Window |
| `make-tile` | width, height, file_path [mask_path] | number, number, [string, [string]] → Tile |
| `make-bitmap-tile` | file_path, [mask_path] | string, [string] → Tile |
| `make-tile-sequence` | tiles | list → Tile-Sequence |
| `generate-mask` | path, colour | string, string → ∅ |

Tabel 1: Procedures beschikbaar na het inladen van de bibliotheek.

## 2.1 Objectgericht Programeren in Scheme

In deze sectie geven we een korte introductie over hoe objectgericht programmeren werkt in Scheme.

Elk van de operaties uit Tabel 1 (met uitzondering van `generate-mask`) geeft een object terug waar je operaties op kan uitvoeren. De grafische bibliotheek is geïmplementeerd in Scheme met behulp van een objectgebaseerde implementatiestijl. Deze stijl maakt het mogelijk om door "berichten te sturen naar objecten van een bepaald ADT" gebruik te maken van diens operaties. Een volledig overzicht van objectgericht programmeren in Scheme (met behulp van dispatch-procedures) kan je terugvinden in het cursusmateriaal van STRUCTUUR VAN COMPUTERPROGRAMMA'S 1. In deze sectie bespreken we kort hoe de operaties die aanwezig zijn op de objecten van de grafische bibliotheek gebruikt kunnen worden.

Een operatie uitvoeren van een van de ADTs van de grafische bibliotheek gebeurt meestal in twee stappen.

In de **eerste stap** stuur je een message (boodschap) naar een ADT. Stel dat `w` een waarde is van het `Window ADT` (met andere woorden: `w` is een object dat een implementatie is van het `Window ADT` zoals beschreven in Sectie 3) dan kunnen we een message naar `w` sturen door `w` uit te voeren (als een procedure) met een bepaald symbool dat de message voorstelt (als argument). Het `Window ADT` heeft een operatie genaamd `set-background!` waarmee de kleur van het venster mee kan aanpassen (zie Tabel 2). Deze operatie kan opgevraagd worden door de `'set-background!` message te sturen naar `w`. Met andere woorden: `(w 'set-background!)`. In dit geval zal `w` een procedure teruggeven waarmee de achtergrond effectief mee aangepast kan worden.

In de **tweede stap** kan vervolgens die procedure opgeroepen worden. De operatie genaamd `set-background!` verwacht 1 argument: namelijk de nieuwe achtergrondkleur van de window. Aangezien `(w 'set-background!)` een procedure teruggeeft kunnen we deze effectief oproepen door er een extra paar haakjes rond deze bestaande expressie te zetten, en vervolgens de argumenten voor die operatie er achteraan aan toe te voegen. Zo zal `((w 'set-background!) "blue")` dus de achtergrondkleur van het venster aanpassen naar blauw.

Merk op dat de tweede stap soms niet nodig is. Operaties die zelf geen argumenten verwachten geven, in plaats van een procedure, onmiddellijk het resultaat terug. Zo zal het `Window ADT` een operatie hebben genaamd `make-layer` (zie Tabel 2) waarmee er een nieuwe laag (het begrip "laag" zal later in dit document besproken worden) mee aangemaakt kan worden. De `make-layer` operatie verwacht zelf geen parameters, dus `(w 'make-layer)` geeft onmiddellijk een waarde terug van het `Layer ADT` in plaats van een procedure die zonder argumenten moet opgeroepen worden `(((w 'make-layer))` zal dus een foutmelding genereren).

[1] We gebruiken rechthoekige haakjes om optionele parameters aan te duiden.

# 3 Window ADT

De constructor om een nieuw venster aan te maken, is de `make-window` procedure. De breedte, de hoogte en de titel van het aan te maken venster worden als argumenten genomen. Er is een vierde, optioneel, argument waarmee het maximum aantal frames per seconde (zie verderop) mee ingesteld kan worden. Indien deze niet meegegeven wordt bij de oproep zal het venster aangemaakt worden met een bovengrens van 60 frames per seconde. Het resultaat van de oproep van `make-window` geeft een nieuw window object terug.

| **Naam** | **Argumenten** | **Signatuur** |
|---|---|---|
| `set-background!` | string | string → ∅ |
| `make-layer` | ∅ | ∅ → Layer |
| `set-key-callback!` | procedure | (symbol, any → ∅) → ∅ |
| `set-update-callback!` | procedure | (number → ∅) → ∅ |

Tabel 2: Window ADT

Tabel 2 geeft een overzicht van de interface van objecten van het `Window ADT`. De `set-background!` boodschap verwacht een string die de kleur van de achtergrond zal veranderen.

Om effectief iets te tekenen op het scherm maakt de grafische bibliotheek gebruik van tiles (en tile sequences, een variant van een tile waarmee eenvoudig animaties mee gemaakt worden). Een tile is een rechthoekig oppervlak waarop pixels op gewijzigd kunnen worden. Tiles kunnen echter niet rechtstreeks toegevoegd worden aan een Window, daarvoor voorziet de grafische bibliotheek de mogelijkheid om lagen aan te maken op een window, waaraan de tiles vervolgens op getekend worden. Door de `make-layer` boodschap te sturen kan de programmeur een nieuwe laag aanmaken. Dit ADT wordt besproken in sectie 4.

De laatste twee boodschappen `set-key-callback!` en `set-update-callback!` zijn de twee procedures die nodig zijn om je spellus te implementeren. Beiden verwachten een procedure als argument.

Voor de `set-key-callback!` procedure is dit argument een procedure verantwoordelijk voor het afhandelen van toetsenbordinvoer. Van deze procedure wordt verwacht dat deze twee parameters heeft: een status en een toets. Wanneer deze procedure uitgevoerd wordt, zal `status` gebonden worden aan ofwel `'pressed` (wanneer de toets ingedrukt wordt), ofwel aan `'released` (wanneer de toets losgelaten wordt). Beiden zijn symbolen. De parameter `key` zal gebonden worden aan een value die overeenkomt met het soort toets dat ingedrukt is. Als dit een lettertoets is dan zal dit gelijk zijn aan een karakterwaarde (bijvoorbeeld `#\a` voor de letter "a", of het symbool `'space` voor de spatie-toets[2]). Merk op dat wanneer een toets voor langere tijd ingedrukt wordt dan kan het zijn dat deze procedure meermaals opgeroepen wordt met `'pressed` voor dezelfde toets.

Voor de `set-update-callback!` procedure is het argument opnieuw een procedure, verantwoordelijk voor het verzorgen van de spellogica. Objecten van het `Window ADT` beschikken zelf over een spellus van waaruit deze procedure bij elke iteratie opgeroepen wordt nadat het window al het tekenwerk afgerond heeft. De meegeven procedure is op die manier verantwoordelijk om de spel- en tekenlogica van het project draaiende te houden. Zelf wordt de procedure opgeroepen met één argument $dt$ (delta time), een number die het aantal milliseconden aangeeft dat verstreken is sinds de vorige oproep van de procedure. Het is met andere woorden de tijd verstreken sinds de vorige iteratie van de spellus.

Door rekening te houden met de "delta time" kan de spellogica zodanig geïmplementeerd worden dat deze altijd voor dezelfde spelervaring zorgt, ongeacht het aantal frames per seconde (aantal iteraties per seconde waarbij de spellogica uitgevoerd wordt en de spellogica op het scherm getekend wordt) waarmee het spel gespeeld wordt. Het is immers niet de bedoeling dat je spelsnelheid anders is op een andere computer (met een hoger of lager aantal frames per seconde). In elk aangemaakt venster zie je ook deze frames per seconde (FPS) verschijnen in de titelbalk (naast de titel die je zelf bepaald hebt).

[2] Lees de documentatie over events in Racket op deze pagina: `https://bit.ly/2EGZ0yr`

Indien dit getal zeer laag wordt, betekent dit dat het uitvoeren van één enkele iteratie van je spellus (= één oproep van het procedure-argument dat aan de procedure `set-update-callback!`) gegeven is, zeer veel tijd in beslag neemt of dat het tekenen door de grafische bibliotheek na het uitvoeren van deze procedure niet optimaal gebeurt (omdat je spel bijvoorbeeld teveel tiles gebruikt, of teveel tiles op 1 laag heeft staan).

# 4 Layer ADT

Omdat je in een spel een achtergrond hebt waarboven objecten getekend moeten worden, voorziet de grafische bibliotheek een concept waarmee de volgorde van wat getekend moet worden op het scherm mee kan bepaald worden: lagen (layers). Elke laag bevat een set van tekenobjecten die zichzelf op de laag kunnen tekenen. Lagen worden door het venster getekend in de volgorde waarin ze aangemaakt werden. Dit betekent dat elk tekenobject van de eerste laag achter de tekenobjecten van de tweede laag zal staan (de volgorde waarbij tiles die op dezelfde laag staan worden getekend, is onbepaald). Bovendien zorgt het gebruik van lagen ervoor dat onderliggende lagen die niet veranderd zijn, niet opnieuw berekend en getekend moeten worden, wat bruikbaar is om de snelheid van het spel optimaal te maken.

| **Naam** | **Argumenten** | **Signatuur** |
|---|---|---|
| `add-drawable` | drawable | Drawable → ∅ |
| `remove-drawable` | drawable | Drawable → ∅ |
| `empty` | ∅ | ∅ → ∅ |
| `draw` | draw-context | DrawContext → ∅ |

Tabel 3: Operaties van het `Layer ADT`.

Tabel 3 geeft de interface van het `Layer ADT` weer. Dit ADT begrijpt de boodschappen `add-drawable` en `remove-drawable` die respectievelijk een tekenelement aan de laag toevoegen en weer verwijderen. Beiden verwachten als argument een value die het type `drawable` hebben. Dit type definieëren we als objecten die minstens de boodschappen `draw` en `set-on-update!` begrijpen. Aan een `draw` boodschap wordt een zogenaamde "drawing context" (aangeduid in Tabel 3 en Tabel 4 als een `DrawContext`) als argument meegegeven. Dit is een intern object van de grafische bibliotheek. Het vereist dus wel wat kennis om zelf een tekenobject te implementeren.

Alle getekende elementen die op een laag getekend zijn kunnen ook in 1 keer verwijderd worden met de `empty` operatie.

Standaard voorziet de grafische bibliotheek al twee soorten `drawables` via de Tile en Tile-Sequence ADTs (zie Sectie 5). Indien je dus zelf geen nieuwe `drawables` toevoegt dan kan je de vermeldingen van `drawable` in de signaturen van Tabel 3 vervangen met `Tile` ⋃ `Tile-Sequence`.

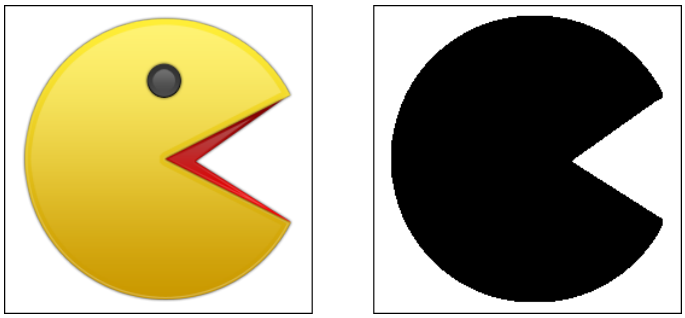
Objecten van het `Layer ADT` begrijpen trouwens ook een `draw` boodschap: deze zal alleen intern gebruikt worden door objecten van het `Window ADT` om de informatie die op een laag staat op het venster getekend te krijgen. Het is doorgaans niet de bedoeling dat jouw project gebruik maakt van deze operatie.

# 5 Tile ADT

De procedure `make-tile` is een constructor die een tile aanmaakt, gegeven een width en height en optioneel een pad naar een bitmap (bv. een png of jpg bestand) en een mask.[3] Een mask is een zwart-wit bitmap die door de bibliotheek gebruikt zal worden om te bepalen welke stukken van de bitmap doorzichtig zijn en welke niet[4].

[3] We voorzien een extra constructor `make-bitmap-tile` die enkel een bitmap neemt en optioneel een mask. Deze constructor gebruikt de hoogte en de breedte van de bitmap voor de achterwege gelaten argumenten.

[4] De bibliotheek heeft geen ondersteuning voor deels-transparante pixels. Grijze pixels in een bitmap zullen genegeerd worden.



Figuur 2: Aan de linkerkant: een bitmap voor Pac-Man. Aan de rechterkant: de mask die hoort bij de bitmap van Pac-Man aan de linkerkant. Merk op dat de vorm gelijk is, en dat de zwarte pixels in de mask overeenkomen met de pixels aan de linkerkant die getekend moeten worden in een Pac-Man spel, de witte pixels in de mask stellen transparante pixels voor.

Figuur 2 toont een voorbeeld van een bitmap voor Pac-Man en diens mask. Indien je geen mask hebt voor je bitmap, kan je procedure `generate-mask` (zie Tabel 1) van de grafische bibliotheek gebruiken om er eentje te genereren. Deze procedure heeft twee argumenten: een pad naar een procedure, en een achtergrondkleur die verdwijnt als de bitmap tile met een mask ingeladen wordt. Bijvoorbeeld `(generate-mask "pacman.pngwhite")` zal een mask genereren (genaamd `pacman_mask.png`) voor de witte pixels in `pacman.png`.

Je bent niet verplicht om masks te gebruiken, maar indien je spelelementen hebt die kunnen overlappen, waarvan de tiles niet rechthoekig zijn, kan je masks gebruiken voor te bepalen welke pixels op de bovenliggende tile transparant moeten zijn.

| **Naam** | **Argumenten** | **Signatuur** |
|---|---|---|
| `set-x!` | x-coördinaat | number → ∅ |
| `set-y!` | y-coördinaat | number → ∅ |
| `get-x` | ∅ | ∅ → number |
| `get-y` | ∅ | ∅ → number |
| `get-w` | ∅ | ∅ → number |
| `get-h` | ∅ | ∅ → number |
| `draw-ellipse` | x,y,width,height, color | number, number, number, number, string → number |
| `draw-rectangle` | x, y, width, height, color | number, number, number, number, string → number |
| `draw-line` | x1, y1, x2, y2, thickness, color | number, number, number, number, number, string → number |
| `draw-text` | text, fontsize, x, y, color | string, number, number, number, string → number |
| `clear` | ∅ | ∅ → ∅ |
| `rotate-clockwise` | ∅ | ∅ → ∅ |
| `rotate-counterclockwise` | ∅ | ∅ → ∅ |
| `draw` | $dc$ | DrawContext → ∅ |
| Enkel voor Tile-Sequences: | | |
| `set-next!` | ∅ | ∅ → ∅ |
| `set-previous!` | ∅ | ∅ → ∅ |

Tabel 4: Operaties van de Tile en Tile-Sequence ADT's.

Tabel 4 bevat een overzicht van alle operaties van het `Tile ADT` (samen met twee operaties die alleen

beschikbaar zijn voor objecten van het `Tile-Sequence ADT`; dit ADT zal apart besproken worden in Sectie 5.1).

Indien je geen gebruik wil maken van een bitmap kan je ook zelf op een tile met operaties zoals `draw-rectangle`, `draw-ellipse`, `draw-text` en `draw-line`. Deze operaties werken ook op tiles die ingeladen zijn met een bitmap, de aanpassingen zullen steeds bovenop de bitmap getekend worden. Hierdoor is het mogelijk om programatisch aanpassingen te maken aan de bitmap. De `clear` operatie maakt de tile leeg, tenzij de tile aangemaakt werd met een bitmap. In dat geval zal `clear` terug de orginele bitmap tonen en alle aanpassingen gemaakt met `draw-rectangle`, `draw-ellipse`, `draw-text` en `draw-line` ongedaan maken.

Een tile kan ook geroteerd worden, hiervoor kunnen de operaties `rotate-clockwise` en `rotate-counterclockwise` gebruikt worden die respectievelijk de tile met de wijzers van de klok, en tegen de wijzers in, zullen roteren.

Om een tile effectief op het scherm te kunnen tekenen moet deze eerst toegevoegd worden aan een laag. Dit kan door gebruik te maken van de `add-drawable` operatie van het `Layer ADT`. Vanaf zodra dit gebeurt is zullen alle daaropvolgende aanpassingen van de tile (bijvoorbeeld diens positie, of aanpassingen aan de tile d.m.v. `clear` of `draw-rectangle`) ook toegepast worden op de laag (en dus ook op de window). Om een tile vervolgens op de juiste plaats te zetten kan er gebruik gemaakt worden met de operaties `set-x!` en `set-y!` (van het `Tile ADT`). Door middel van `get-x` en `get-y` kan de huidige positie opgevraagd worden en met `get-w` en `get-h` kan respectievelijk de breedte en de hoogte van de tile mee opgevraagd worden.

## 5.1 Tile-Sequence ADT

Met `make-tile-sequence` maak je een Tile-Sequence aan. Een Tile-Sequence is een aaneenschakeling van Tiles waar je sequentieel door kan lopen, de constructor `make-tile-sequence` verwacht dan ook een lijst van Tiles als argument. Deze abstractie is vooral handig om animaties in je spel te incorporeren.

Net zoals een Tile zal een Tile-Sequence een enkele bitmap op het venster tekenen, maar je kan via oproepen van `set-next!` en `set-previous!` snel de volgende of vorige tile in de sequentie afbeelden. De signaturen van deze operaties kunnen ook teruggevonden worden in Tabel 4.

Bijvoorbeeld, in een Pac-Man spel zou je Tile-Sequences kunnen gebruiken om een animatie te maken waarbij Pac-Man hapt. Dit doe je door een Tile-Sequence aan te maken met twee Tiles. De eerste Tile maak je aan voor een bitmap van Pac-Man waarop de mond gesloten is, en de tweede Tile voor een bitmap waar de mond van Pac-Man open is. Als je dan snel achter elkaar `set-next!` oproept, zal je Pac-Man lijken te happen.

Een Tile-Sequence ondersteunt dezelfde tekenoperaties als een enkele Tile. Deze kunnen ook teruggevonden worden in Tabel 4. Hiertoe worden deze operaties gedelegeerd naar de interne lijst van Tiles die bijgehouden wordt door de objecten van het `Tile-Sequence ADT`. Indien je tekent op een Tile-Sequence, wordt dit op alle Tiles in de sequence getekend. Echter, wanneer je rechtstreeks tekent op een Tile van een Tile-Sequence, zal het resultaat alleen zichtbaar zijn wanneer de Tile-Sequence toevallig net die Tile aan het afbeelden is. Merk op wanneer zowel een Tile uit een Tile-Sequence als de Tile-Sequence zelf aan een laag zijn toegevoegd de grafische bibliotheek perongeluk de foute tile kan weghalen wanneer een Tile-Sequence naar een volgende (of vorige) tile gaat (objecten van het Tile-Sequence ADT gaan immers bij elke transitie naar een andere tile in de sequence, de oude tile weghalen en de nieuwe tile toevoegen aan de laag): voeg dus alleen de Tile-Sequence zelf toe aan een laag om bugs te vermijden.