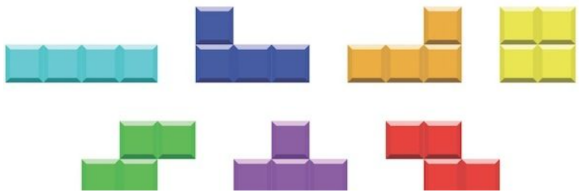
Opgavebesvarelse 10g

Shatin Nguyen (hlv332), Peter Asp Hansen (glt832) & Oliver Fontaine Raaschou (vns328)

23. februar 2023

I denne rapport, vil vi gennemgå vores kode til vores simple version af Tetris. Vores spil skal fungere på følgende måde:

1. Spillet starter med at en tilfældig brik placeres i toppen af vores board. Udvalget af brikker ses i billedet nedenfor



Figur 1: Tetris brikker

   I koden betegner vi disse brikker vha typen `bool[,]`. Yderligere er hver brik navngivet på følgende måde: `straight()`, `l(false)`, `l(true)`, `square()`, `z(false)`, `t()`, `z(true)`, hvor true og false for l og z brikkerne viser, om brikkerne er spejlvendt.

2. En brik er i bevægelse af gangen og bevæges ved at brugeren trykker på piltasterne. Vi har valgt at implementere det således at 'op' roterer en brik med 90 grader i urets retningen, 'højre' bevæger en brik et felt mod højre og et felt ned, og 'venstre' bevæger en brik en mod venstre og en nedad. 'ned' bevæger en brik en ned.

3. Brikkerne i vores spil skal ikke kunne overlappe ind i hinanden.

4. Måden vi får nye brikker frem er når en aktiv brik ikke længere har nogle mulighed for at rykke sig, så kommer der et nyt frem i toppen.

5. Spillet slutter så snart at en ny brik ikke kan komme frem i toppen.

6. Denne implementation af Tetris vil scoring være mængden af brikker på brættet, når spillet er slut.

10g0

I første del af opgaven skal vi lave en Canvas draw funktion: `draw: w:int -> h:int -> s:state -> Canvas.canvas`, som opdeler et 2D array af størrelsen $10x20$ i et canvas vindue $300x600$. For at 2D arrayet skal udfylde canvas vinduet, så definerer vi nogle hjælpe variabler `y` som er højden af vinduet dividere med højden af boardet, og `x` som er længden af vinduet dividere med længden af boardet. Til sidst skal `draw` blot itererer igennem 2D arrayet, og hvis der er en farve, altså `Some e`, så tegnes farven på dens position fra $(x \cdot x', y \cdot y')$ til $(x \cdot (x'+1), y \cdot (y'+1))$, hvor x' og y' er indeks værdierne i 2D arrayet.

```
let draw (w: int) (h: int) (s: state) =
    let C = Canvas.create w h
    let y = h/s.height
    let x = w/s.width
    s.board |> Array2D.iteri(fun x' y' e -> if e.IsSome then do setFillBox C (color e) (x
    *x',y*y')(x*(x'+1),y*(y'+1)))

```

10g1

Denne del af opgaven skal vi implementere klassen `tetromino`. Klassens `body constructors`, er der tre `let` bindinger, som blot er værdierne fra `header`. Disse laves til som `mutable`, så de kan ændre værdi. I vores klasses `property` sektion, laver vi `this.image`, `this.col`, `this.offset`, `this.height` og `this.width` som returnere værdierne beskrevet i opgavebeskrivelsen. I klassens `method` sektion, laver vi `this.rotateRight()`. Måden `rotateRight()` fungere er, at den bytter om på højden og længden af en brik, og farverne flyttes ved at bytte rundt på deres position x og y position, hvor den nye x positon er $w-1-j$, hvor w er længden af brikken og j i koden er den før rotationens y værdi. For at vi kan printet outputtet fra `tetromino`, har vi lavet `override this.ToString()`.

```
type tetromino (a:bool[,], c:Color, o:position) =
    let mutable _image = a
    let mutable _color = c
    let mutable _offset = o

    member this.image
        with get() = _image
        and set (a) = _image <- a
    member this.col = _color
    member this.offset
        with get() = _offset
        and set (a) = _offset <- a
    member this.height = Array2D.length2 this.image
    member this.width = Array2D.length1 this.image

    member this.rotateRight() =
        let h = this.height
        let w = this.width
        Array2D.init this.height this.width (fun i j -> this.image[w-1-j,i])

    override this.ToString() = sprintf "%A" (this.image)
```

Når vi definerer vores brikker, bruger vi `inherit` til at lave en afledte klasse af `tetromino`. Her betegner vi formen af vores brik vha. et 2d array. Eksempel: t defineres ved at lave et 3x2 array med boolean værdi på true, hvorefter vi ændrer boolean værdien på plads `[0,0]` og `[2,0]` til false ved at skrive `base.image[a,b] <- false`. Yderligere så har vi også defineret farven af en brik, `c`, og offset, `o`, som beskriver startpositionen. Vi udnytter i `react` at vi kan ændre offset ved at trykke på piltasterne. I vores implementation af rotateright roterer vi mod uret, i stedet for med uret. En anden lille udfordringer er når en brik er placeret langs højre side af vores board og den roteres, kan vi risikere at brikken i få tilfælde er 'out of bounds' og spillet dermed crasher. Dette skyldes at der ikke er taget højde for rotation i vores `isValid` metode.

```
type square() =
    inherit tetromino((Array2D.create 2 2 true),Yellow,(4,0))

type straight() =t
    inherit tetromino((Array2D.create 4 1 true), Cyan, (3,0))

type t1()=
    inherit tetromino((Array2D.create 3 2 true), Red, (3,0))
    do
        base.image[0,0] <- false
        base.image[2,0] <- false

type l(mirror:bool) =
    inherit tetromino((Array2D.create 3 2 true),(if mirror then Orange else Blue),(3,0))
    do
        if mirror then
            do
                base.image[0,0] <- false
                base.image[1,0] <- false
        else
            do
                base.image[1,0] <- false
                base.image[2,0] <- false

type z (mirror:bool) =
    inherit tetromino((Array2D.create 3 2 true),(if mirror then Green else Red),(3,0))
    do
        if mirror then
            do
                base.image[0,0] <- false
                base.image[2,1] <- false
        else
            do
                base.image[2,0] <- false
                base.image[0,1] <- false
```

Figur 2: Shapes

10g2

Vi laver en klasse kaldet `board`. Denne klasse indeholder en række metoder der skal hjælpe med at bevæge brikkerne på vores Canvas.

```
type board (w:int, h:int) =
    let _board = Array2D.create w h None
    let mutable (t:tetromino option) = None
    let isOccupied(t:tetromino, newo) : bool =
        let (x,y) = newo
        let mutable occupied = false
        for i = x to (Array2D.length1(t.image)-1+x) do
            for j = y to (Array2D.length2(t.image)-1+y) do
                if (t.image[i-x,j-y]=true) && _board[i,j].IsSome then
                    occupied <- true
        occupied

    member this.width = w
    member this.height = h
    member this.board : Color option[,] = _board
    member this.current
        with get()= t
        and set(a) = t <- a

    member this.newPiece() : tetromino option =
        let rnd = System.Random()
        let piece =
            match rnd.Next(6) with
                |0 -> (straight()):> tetromino
                |1 -> (square()) :> tetromino
                |2 -> (t1()) :> tetromino
                |3 -> (l(true)):> tetromino
                |4 -> (l(false)):> tetromino
                |5 -> (z(true)):> tetromino
                |_ -> (z(false)):> tetromino
        Some piece

    member this.put (t:tetromino) : bool =
        let (a,b) = t.offset
        Array2D.iteri (fun i j image -> if (_board[i,j].IsSome) then () elif image = true
     then _board[a+i,b+j] <- Some t.col) t.image
        this.current <- Some t
        true

    member this.take() : tetromino option =
        let mutable a = Option.get(this.current)
        let (x,y) = a.offset
        Array2D.iteri (fun i j image -> if image=true then _board[x+i,y+j] <- None) a.
    image
        Some a

    member this.isValid(t:tetromino,move:position): bool =
        let (x,y) = t.offset
        let mutable valid = false
        let newo = (x+(fst move),y+(snd move))
        if  (w - t.width + 1 > (fst newo)) && (h - t.height + 1 > (snd newo)) && ((fst
    newo) >= 0) && ((snd newo)>= 0)  then
            if isOccupied(t,newo) then
                valid <- false
            else
                valid <- true
        else
            valid <- false
        valid
```

Det kan ses i koden, at klassen `board` indeholder følgende metoder. De er noteret og forklaret lige nedenfor.

- this.width er bredden på vores board i integer format
- this.height er højden på vores board i integer format

- this.board bruges til at holde øje med selve boardet så det kan muteres.
- this.newPiece() returnerer et nyt piece som bl.a kan være straight, z og square.
- this.current bruges til at holde øje med det nuværende piece
- this.put(t) sætter vores tetromino t på brættet. Idéen med denne metode var at den skal returnere en boolean statement som fortæller om brikken kan placeres. Vi har desværre været nødsaget til altid at returnere true, hvilket gør at spillet ikke afsluttes når vi når toppen og dermed ikke kan put længere.
- this.take() tager den nuværende (current) tetrimino og undersøger hvorledes den kan placeres.
- this.isValid(t,move) bruges til at tjekke om brikkens (t) næste skridt kan udføres.Det gør den ved at undersøge om brikken bl.a er indenfor boardet eller om pladsen allerede er taget.

Vi benytter hjælpefunktionen `isOccupied(t,newo)` i metoden `isValid(t,move)` til at undersøge om pladsen allerede er optaget af en anden brik.

10g3

Vi laver en `react` funktionen som tager vores state og en Canvas.key som input. Dermed kan vi angive bevægelsen af brikkerne med piltasterne.

```
let react (s:state) (k:Canvas.key) : state option =
    let checkIsDead(x) =
        if (s.isValid(x,(1,1))) || (s.isValid(x,(-1,1))) || (s.isValid(x,(0,1))) then
            false
        else
            true
    let mutable isDead = false
    let endPiece(x:bool) =
        if x=true then
            s.newPiece() |> Option.get |> s.put |> ignore
            Some s
        else
            Some s
    let x = Option.get(s.take())
    match getKey k with
        |LeftArrow ->
            let (x1,y1) = x.offset
            if s.isValid(x,(-1,1)) then
                x.offset <- (x1-1,y1+1)
                isDead <- checkIsDead(x)
                s.put(x)
            else
                isDead <- checkIsDead(x)
                s.put(x)
            endPiece(isDead)
        | RightArrow ->
            let (x1,y1) = x.offset
            if s.isValid(x,(1,1)) then
                x.offset <- (x1+1,y1+1)
                isDead <- checkIsDead(x)
                s.put(x)
            else
                isDead <- checkIsDead(x)
                s.put(x)
            endPiece(isDead)
        | UpArrow ->
            x.image <- x.rotateRight()
            s.put(x)
            checkIsDead(x)
            Some s
        | DownArrow ->
            let (x1,y1) = x.offset
            if s.isValid(x,(0,1)) then
                x.offset <- (x1,y1+1)
                isDead <- checkIsDead(x)
                s.put(x)
            else
                isDead <- checkIsDead(x)
                s.put(x)
            endPiece(isDead)
        |_-> None

let b = board(10,20)
b.newPiece()|> Option.get |> b.put
runApp "Tetris" 300 600 draw react b
```

I react funktion bruges der en hjælpefunktion kaldet `checkIsDead(x)`. Denne funktion skal vi bruge til at vurdere hvornår en brik er død og der dermed skal genereres en ny. Det er bl.a. her der er plads til forbedringer. Vi oplever i hvert fald at når vores current tetromino lander på en brik som tidligere er blevet placeret, opstår der en bug hvor en brik ikke kan sættes. Her er vi i stedet tvunget til at bevæge vore brik til siderne netop fordi `checkIsDead(x)` returnere false i stedet for det ønskede true. Bemærk at dette er et særtilfælde og derfor har vi valgt at ignorere problemet, selvom den optimale løsning selvfølgelig ville have været hvis vi kunne tage højde for dette.
Vi har brugt pattern matching i `react` til at definere de tilhørende handlinger til hver key. Betragt f.eks. input key `LeftArrow`. Her bruger vi et if-statement til at tjekke om brikken kan bevæge sig en til venstre og en ned vha. `isValid(x,move)` metoden. Hvis dette er sandt fortager vi følgende handlinger.

- vi ændrer offset med `move`
- vi ændrer den mutable værdi isDead så den er magen til `checkIsDead(x)`

- vi placerer brikken på den opdaterede position vha. `put(x)`

Hvis dette er `isValid(x,move)` er falsk, gør vi det samme med undtagelse af det første skridt. Yderligere så kaldes `endPiece(isDead)` udenfor vores if-statement. `endPiece(x)` er en hjælpefunktion som hjælper os med at placere en ny brik hvis `isDead` er sand. Bemærk at `isDead` oprindeligt har værdien falsk.

10g4

Skipped

10g5

Følgende delopgave indeholder henholdsvis en black-box test og en white-box test for de to klasser, `tetromino` og `board`.
Først betragtes en black-box test for `tetromino`. Til at starte med, laver vi en `tetromino` prik, hvilket vi i vores tilfælde vil kaldes `tb`. Vores test `tetromino`, er en gul $2x2$ prik, placeret i positionen $(4, 0)$. Her benytter vi efterfølgende en print statement til at tjekke om, hver `property` og `method` stemmer overens med vores forventet resultat. Her burde outputtet være true i alle tilfælde, hvilket det gør

```
//Black box test af tetromino
let tb = tetromino((Array2D.create 2 2 true),Yellow,(4,0))
printfn "TB 1: %A" (tb.image = (Array2D.create 2 2 true))
printfn "TB 2: %A" (tb.col = Yellow)
printfn "TB 3: %A" (tb.offset = (4, 0))
printfn "TB 4: %A" (tb.height = 2)
printfn "TB 5: %A" (tb.width = 2)
printfn "TB 6: %A" (tb.rotateRight() = tb.image)
```

For at teste de forskellige `properties` og `method` i vores `board`, anvender vi samme metode. Vi laver et `board` kaldet $tb2$, i dette tilfælde på $35x30$, hvor de tre `members`: `width`, `height` og `board`, nemt kan testet, ved at skrive lig med den boards headers. Vi forventer at en længden er 35, højden er 30 og det er et tomt board på $35x30$. For at teste `this.newPiece()`, forsøgte vi først at kalde `newPiece()` og sige lig med de forskellige prikker. Vi stødte ind i et problem med, hvergang vi kaldte `newPiece()` og tjekkede om den var en af vores prikker, så genererede `newPiece()` en ny prik ved hvert tjek. Vi at forsøgte `printfn "%A"(tb2.newPiece() = Some (square()) || tb2.newPiece() = Some (straight)...)`, hvilket i nogle tilfælde ville returnere `false`. Vi valgte derfor en ny metode, hvor vi lavede en funktion som ville putte `newPiece()` til et nyt board `a` og hvis `newPiece()` er lig med vores aktive prik i det nye board, så returneres `true` ellers `false`. Funktionen returnere også `false`, hvis det nye board `a` er tomt. Dette er formålet med funktion `tb2_np()`.

De næste tre `members`: `this.current`, `this.put()` og `this.isValid()` returnere allerede en `bool` og vi printer dem blot. Den sidste `property` `this.take()`, skal blot være lig med test-boards `this.current` og er de to lig med hinanden, så virker `this.take()`. Herunder er vores black box test af klassen `board`.

```
//Black box test af board
let tb2 = board(35,30)
printfn "TB 1: %A" (tb2.width = 35)
printfn "TB 2: %A" (tb2.height = 30)
printfn "TB 3: %A" (tb2.board = Array2D.create 35 30 None)

let tb2_np() =
    let a = board(10,10)
    let mutable i = false
    let p = a.newPiece()
    match p with
        |Some x ->
                a.put(x)
                if a.board = (Array2D.create 10 10 None) then
                    i <- false
                elif a.take() = p then
                    i <- true
                else
                    i <- false
                i
        | None -> false
printfn "TB 4: %A" (tb2_np())
printfn "TB 5: %A" (tb2.current = None)
printfn "TB 6: %A" (tb2.put(square()))
printfn "TB 7: %A" (tb2.take() = tb2.current)
printfn "TB 8: %A" (tb2.isValid(square(), (1,1)))
```

For at lave en white box test af `tetrimono` og `board`, laver vi en liste som indeholder to elementer. I indeks 0 er vores funktionskald og i indeks 1 er vores forventede resultater. De forventede resultater er de samme fra vores black box test. Vi laver derefter en funktion som tjekker om indeks 0 er lig med indeks 1 i listen, og hvis `true`, så ved vi at alle metoderne vikker og ved `false`, så virker minimum en af metoderne ikke. Herunder er vores white box test.

```
//White box test af tetrimono
let tw = tetromino((Array2D.create 2 2 true),Yellow,(4,0))
let wb_test_set_tetromino = [
    [tw.image, tw.col, tw.offset, tw.height, tw.width, tw.rotateRight()];
    [(Array2D.create 2 2 true), Yellow, (4,0), 2, 2, (Array2D.create 2 2 true)]
    ]

let test_wb() =
    if wb_test_set_tetromino[0] = wb_test_set_tetromino[1] then
        printfn "Alle metoder virker"
    else
        printfn "Minimum 1 metode virker ikke"
printfn "%A" (test_wb())
```

```
//White box test af board
let tw2 = board(35,30)
let wb_test_set_board = [
    [tw2.width, tw2.height, tw2.board, tb2_np() ,tw2.current, tw2.put(square()), tw2.take
    (), tw2.isValid((square(),(1,1)))];
    [35, 30, (Array2D.create 35 30 None), true , None, true, tw2.current, true]
    ]

let test_wb2() =
    if wb_test_set_tetromino[0] = wb_test_set_tetromino[1] then
        printfn "Alle metoder virker"
    else
        printfn "Minimum 1 metode virker ikke"
printfn "%A" (test_wb2())
```