Programmering og Problemløsning

13.2: Objektorienteret design og UML

Design efter navne- og udsagnsord

Navneord er kandidater til klasser, udsagnsord er kandidater til metoder, som får klasserne til at hænge sammen.

Navneord (substantiv): Genstande, levende væsner, personer eller abstraktioner.

F.eks.: (et) bord, (en) mus, Jon, (et) venskab.

Udsagnsord (verbum): Handlinger, hændelser, tilstandsmåder.

F.eks.: (at) løbe, (han) underviser, (han) er (lærer)

Design efter navne- og udsagnsord

Use-case: Sænke slagskibe

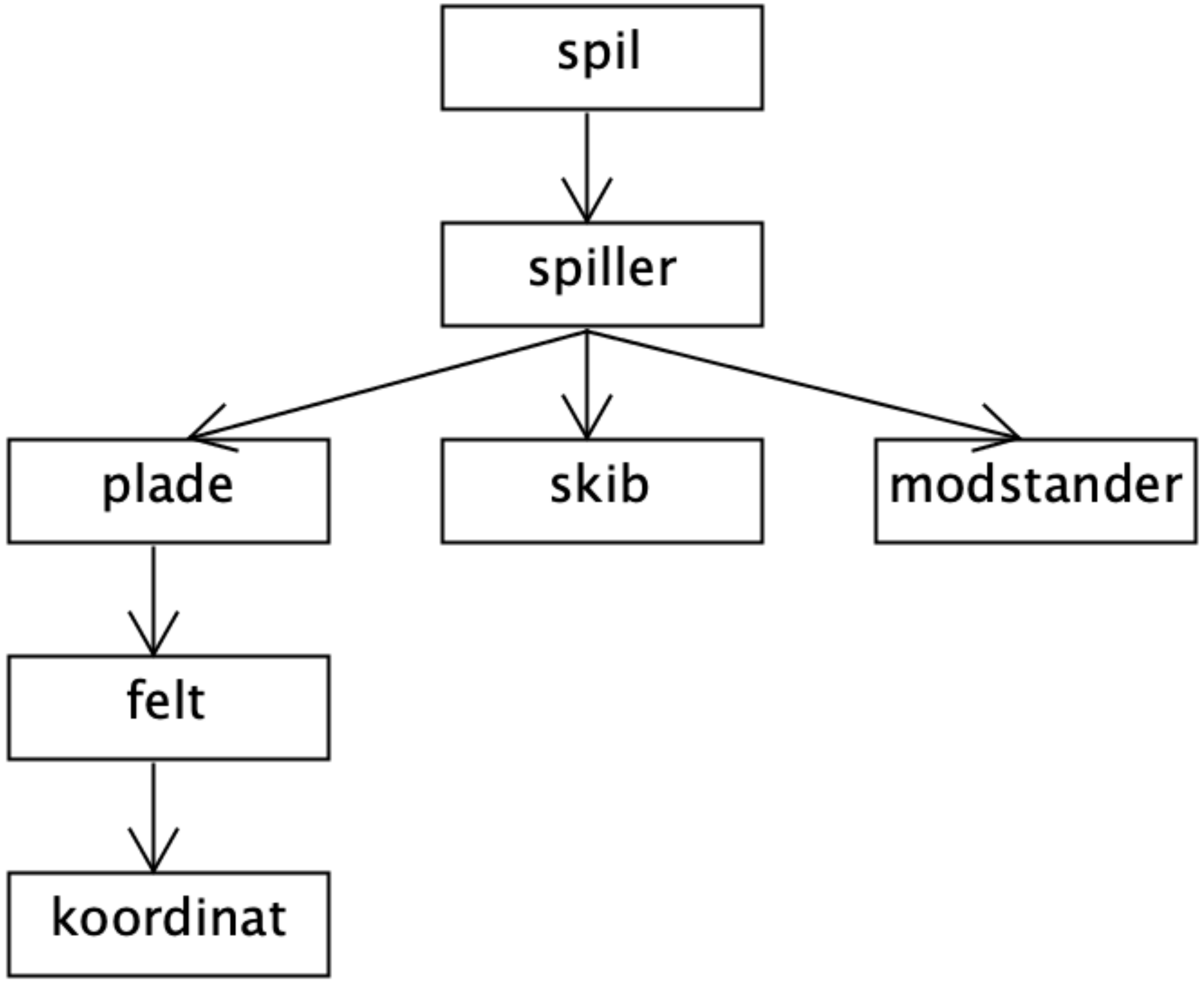
Dette er et spil for to personer, der kan spilles med papir og blyant. Der spilles på fire plader, to for hver spiller, og hver plade er inddelt i 10x10 felter. Hvert felt identificeres vha. dets række- og søjlenummer.

Hver spiller får tildelt et antal skibe, som placeres på spillerens ene plade og markerer, hvor modstanderen har forsøgt at skyde. På den anden plade markerer spilleren tilsvarende, hvor han/hun har forsøgt at ramme modstanderen.

Når skibene er placeret skiftes spillerne til at skyde på modstanderens felt, og modstanderen annoncerer ramt eller plask, alt efter om et skib blev ramt eller ej. Vinderen er den, der først får sænket alle modstanderes skibe.

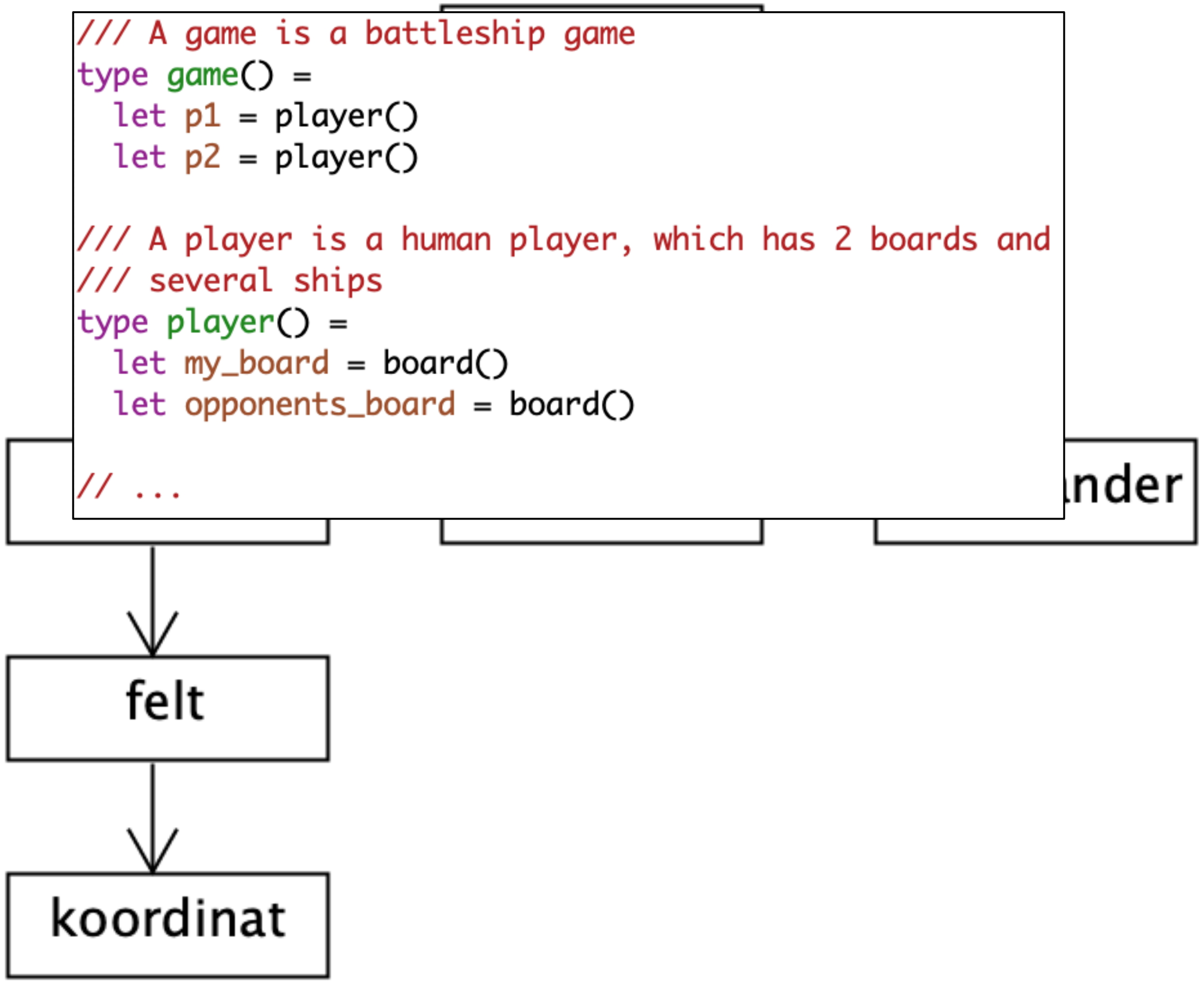
Sænke slagskibe

```
/// A game is a battleship game
type game() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() = class end

/// An opponent is another player
type opponent() = class end

/// A ship can be damaged
type ship() = class end

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() = class end

/// A field is can be covered by a ship and can have been
/// shootend at
type field() = class end

/// A coordinate is a location on a board
type coordinate() = class end
```



Sænke slagskibe

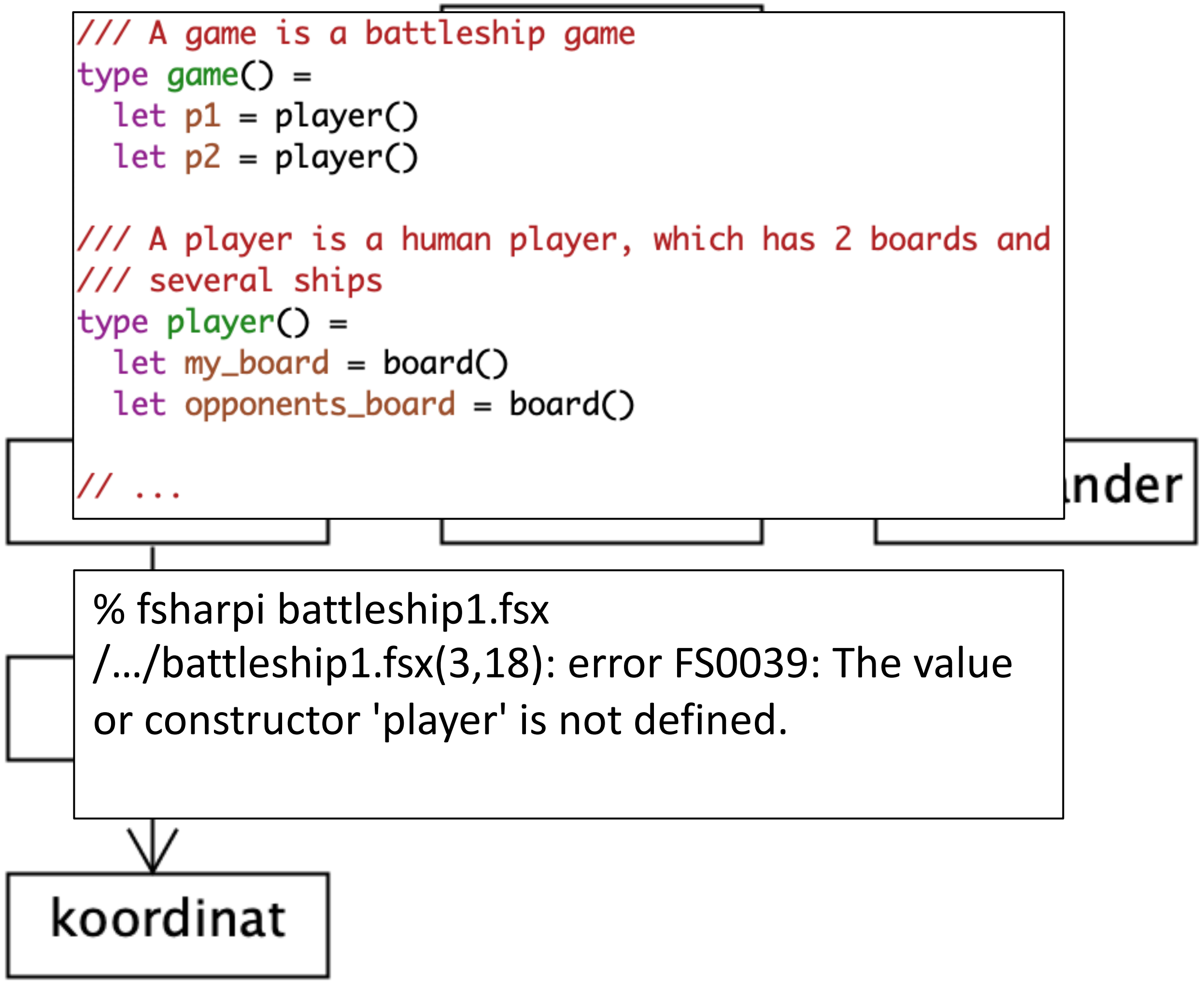
```
/// A game is a battleship game
type game() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() = class end

/// An opponent is another player
type opponent() = class end

/// A ship can be damaged
type ship() = class end

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() = class end

/// A field is can be covered by a ship and can have been
/// shootend at
type field() = class end

/// A coordinate is a location on a board
type coordinate() = class end
```



```
/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()

// ...
```

Sænke slagskibe

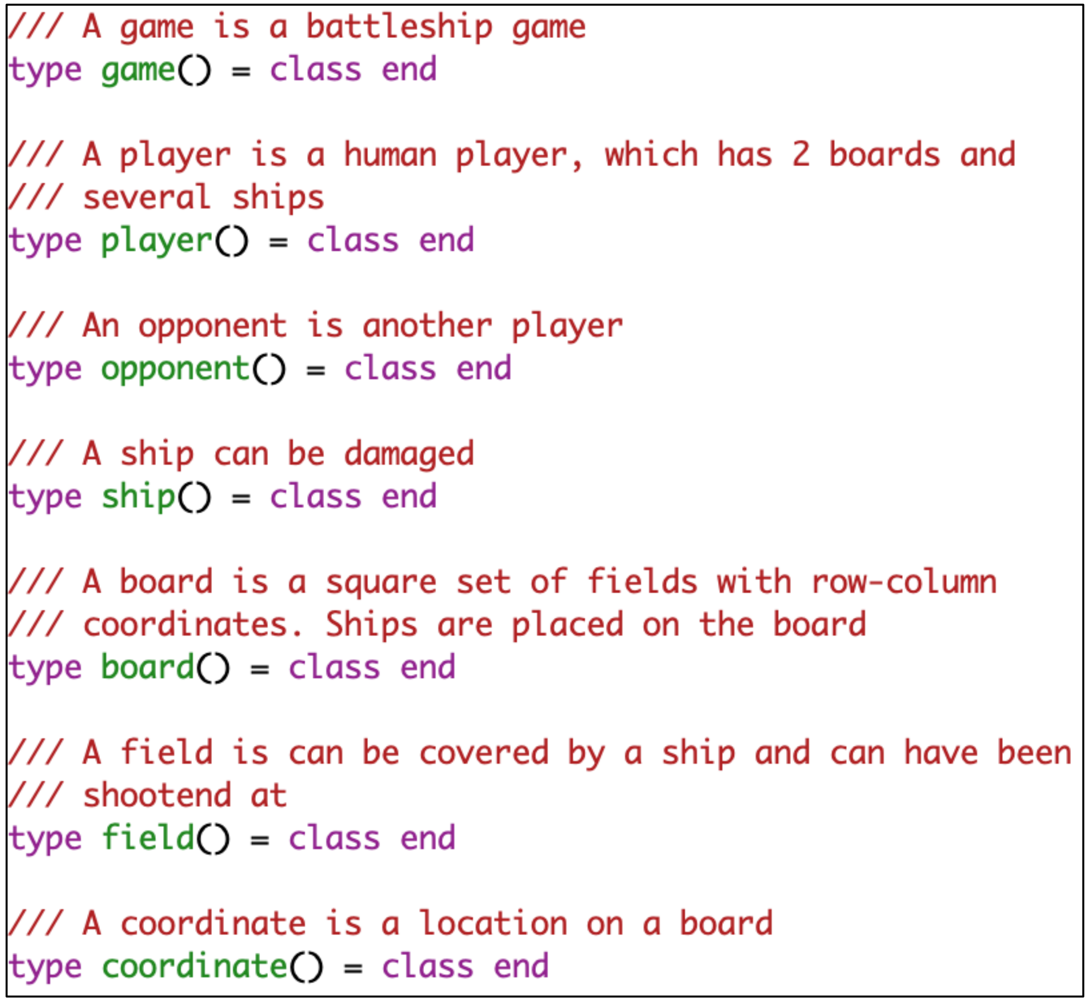
```
/// A game is a battleship game
type game() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() = class end

/// An opponent is another player
type opponent() = class end

/// A ship can be damaged
type ship() = class end

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() = class end

/// A field is can be covered by a ship and can have been
/// shootend at
type field() = class end

/// A coordinate is a location on a board
type coordinate() = class end
```



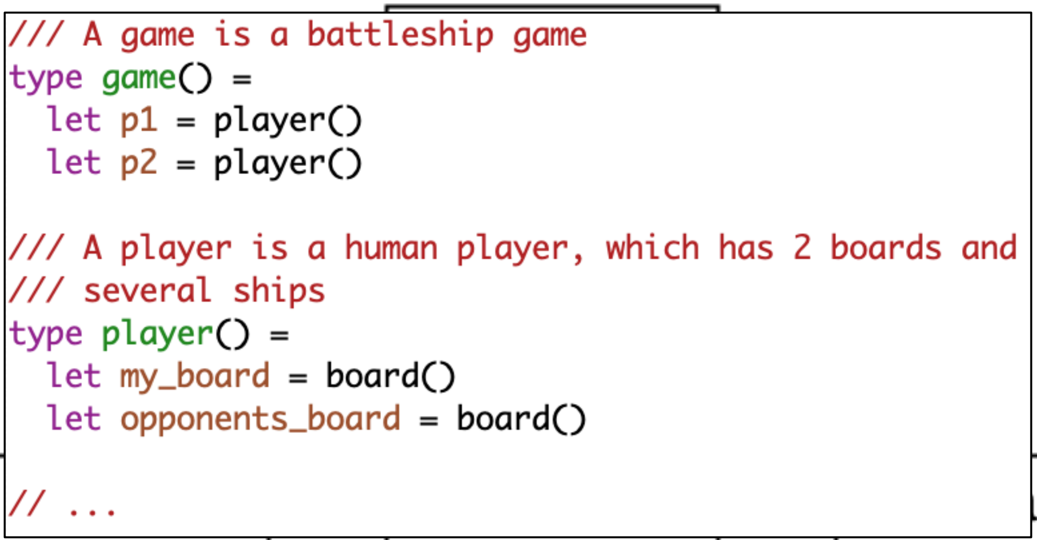
```
/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()

// ...
```

```
% fsharpi battleship1.fsx
/.../battleship1.fsx(3,18): error FS0039: The value
or constructor 'player' is not defined.
```

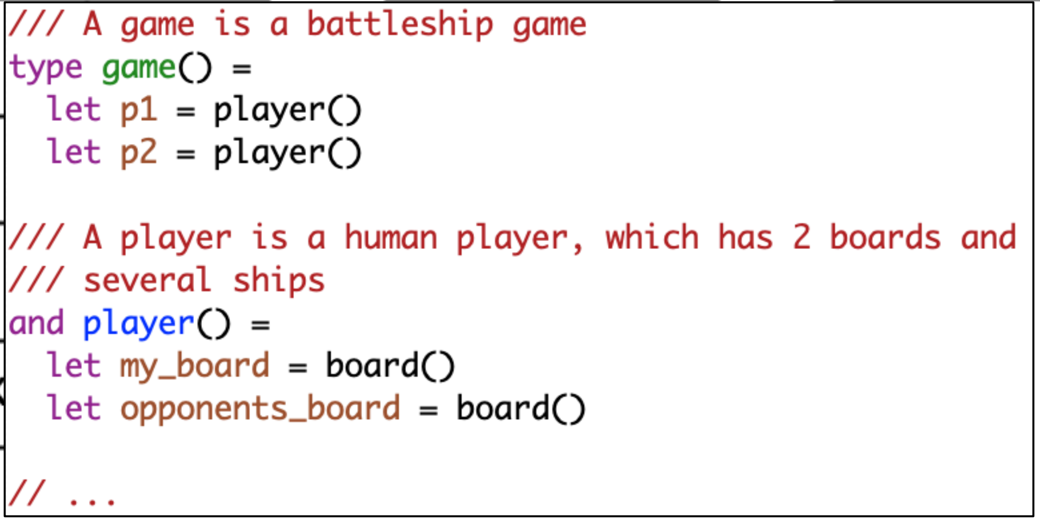
Sænke slagskibe



```
/// A game is a battleship game
type game() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() = class end

/// An opponent is another player
type opponent() = class end

/// A ship can be damaged
type ship() = class end

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() = class end

/// A field is can be covered by a ship and can have been
/// shootend at
type field() = class end

/// A coordinate is a location on a board
type coordinate() = class end
```



```
/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()

// ...
```



```
/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()

/// A player is a human player, which has 2 boards and
/// several ships
and player() =
  let my_board = board()
  let opponents_board = board()

// ...
```

Sænke slagskibe

```fsharp
/// A game is a battleship game
type game() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() = class end

/// An opponent is another player
type opponent() = class end

/// A ship can be damaged
type ship() = class end

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() = class end

/// A field is can be covered by a ship and can have been
/// shootend at
type field() = class end

/// A coordinate is a location on a board
type coordinate() = class end
```

```fsharp
/// A coordinate is a location on a board
type coordinate(row : int, col : int) =
  member this.row with get () = row
  member this.col with get () = col

/// A field is can be covered by a ship and can have been
/// shootend at
type field(c : coordinate) =
  member this.coord with get () = c

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> field (coordinate (i, j)))

/// A ship can be damaged
type ship() = class end

/// An opponent is another player
type opponent() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()

/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()
```

```
/// A coordinate is a location on a board
type coordinate(row : int, col : int) =
  member this.row with get () = row
  member this.col with get () = col

/// A field is can be covered by a ship and can have been
/// shootend at
type field(c : coordinate) =
  member this.coord with get () = c

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> field (coordinate (i, j)))

/// A ship can be damaged
type ship() = class end

/// An opponent is another player
type opponent() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()

/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()
```

Sænke slagskibe

Træd et skridt tilbage:

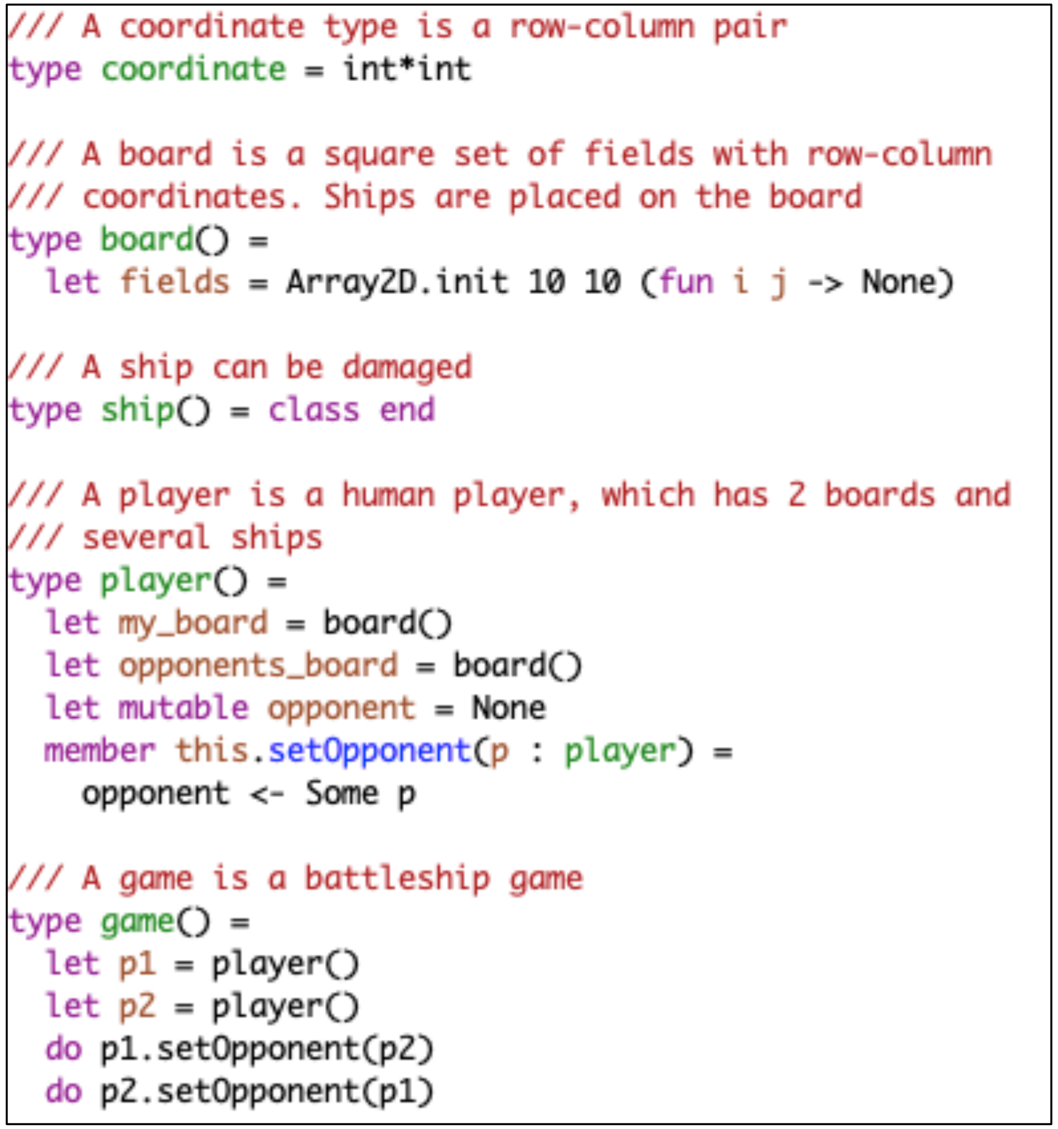
- Koordinater er bare heltalspar
- Index i et array er en koordinat
- Arrays har felter
- En modstander er også en spiller.

Sænke slagskibe

```
/// A coordinate is a location on a board
type coordinate(row : int, col : int) =
  member this.row with get () = row
  member this.col with get () = col

/// A field is can be covered by a ship and can have been
/// shootend at
type field(c : coordinate) =
  member this.coord with get () = c

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> field (coordinate (i, j)))

/// A ship can be damaged
type ship() = class end

/// An opponent is another player
type opponent() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()

/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()
```

```
/// A coordinate type is a row-column pair
type coordinate = int*int

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> None)

/// A ship can be damaged
type ship() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()
  let mutable opponent = None
  member this.setOpponent(p : player) =
    opponent <- Some p

/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()
  do p1.setOpponent(p2)
  do p2.setOpponent(p1)
```

Sænke slagskibe



```
/// A coordinate type is a row-column pair
type coordinate = int*int

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> None)

/// A ship can be damaged
type ship() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()
  let mutable opponent = None
  member this.setOpponent(p : player) =
    opponent <- Some p

/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()
  do p1.setOpponent(p2)
  do p2.setOpponent(p1)
```

~~Et spil består af spillere.~~

~~En spiller har en modstander.~~

En spiller har skibe.

~~En spiller har plader.~~

~~En plade består af felter.~~

~~Et felt har et koordinat.~~

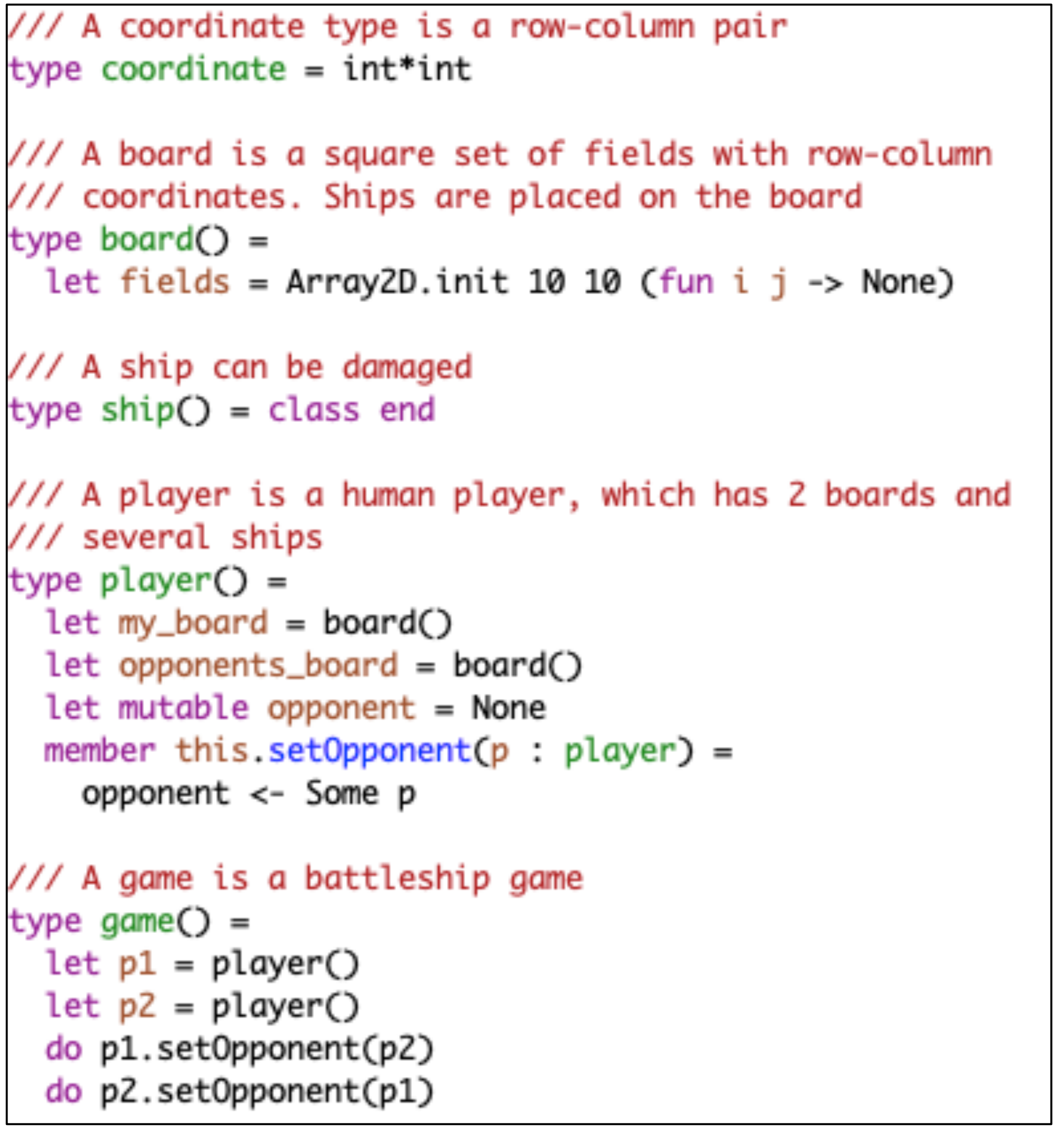
En spiller kan vinde.

En spiller skyder på en modstander.

En spiller markerer skud på en plade.

Et skib kan blive ramt.

Sænke slagskibe



```
/// A coordinate type is a row-column pair
type coordinate = int*int

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> None)

/// A ship can be damaged
type ship() = class end

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()
  let mutable opponent = None
  member this.setOpponent(p : player) =
    opponent <- Some p

/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()
  do p1.setOpponent(p2)
  do p2.setOpponent(p1)
```

~~Et spil består af spillere.~~

~~En spiller har en modstander.~~

En spiller har skibe.

~~En spiller har plader.~~

~~En plade består af felter.~~

~~Et felt har et koordinat.~~

En spiller kan vinde.

En spiller skyder på en modstander.

En spiller markerer skud på en plade.

Et skib kan blive ramt.

Opgaven er ikke fuldt specificeret, hvad gør vi?

Sænke slagskibe

Opgaven er ikke fuldt specificeret, hvad gør vi?

- Identificer uklare eller løse specifikationer
- Beskriv mulige løsninger
- Vælg og argumentér

Use-case: Sænke slagskibe

Dette er et spil for to personer, der kan spilles med papir og blyant. Der spilles på fire plader, to for hver spiller, og hver plade er inddelt i 10x10 felter. Hvert felt identificeres vha. dets række og søjle nummer.

Hver spiller får tildelt et antal skibe, som placeres på spillerens ene plade og markerer, hvor modstanderen har forsøgt at skyde. På den anden plade markerer spilleren tilsvarende, hvor han/hun har forsøgt at ramme modstanderen.

Når skibene er placeret skiftes spillerne til at skyde på modstanderens felt, og modstanderen annoncerer ramt eller plask, alt efter om et skib blev ramt eller ej. Vinderen er den, der først får sænket alle modstanderes skibe.

Sænke slagskibe

Opgaven er ikke fuldt specificeret, hvad gør vi?

- Identificer uklare eller løse specifikationer
- Beskriv mulige løsninger
- Vælg og argumentér

F.eks.:

- Opgaven specificerer ikke, hvor mange skibe, der skal være i spillet, ej heller hvilken form de skal have.
- Typiske spil har 4-5 skibe, som er lige, ligger enten horisontalt eller verticalt, og har størrelse mellem 2 og 5 felter.
- Vi vælger 4 skibe, 2 små, 1 mellem og 1 stort. Vi sikrer at programmet nemt kan opdateres med et andet antal og størrelser.

Use-case: Sænke slagskibe

Dette er et spil for to personer, der kan spilles med papir og blyant. Der spilles på fire plader, to for hver spiller, og hver plade er inddelt i 10x10 felter. Hvert felt identificeres vha. dets række og søjle nummer.

Hver spiller får tildelt et antal skibe, som placeres på spillerens ene plade og markerer, hvor modstanderen har forsøgt at skyde. På den anden plade markerer spilleren tilsvarende, hvor han/hun har forsøgt at ramme modstanderen.

Når skibene er placeret skiftes spillerne til at skyde på modstanderens felt, og modstanderen annoncerer ramt eller plask, alt efter om et skib blev ramt eller ej. Vinderen er den, der først får sænket alle modstanderes skibe.

Sænke slagskibe

Skibe:

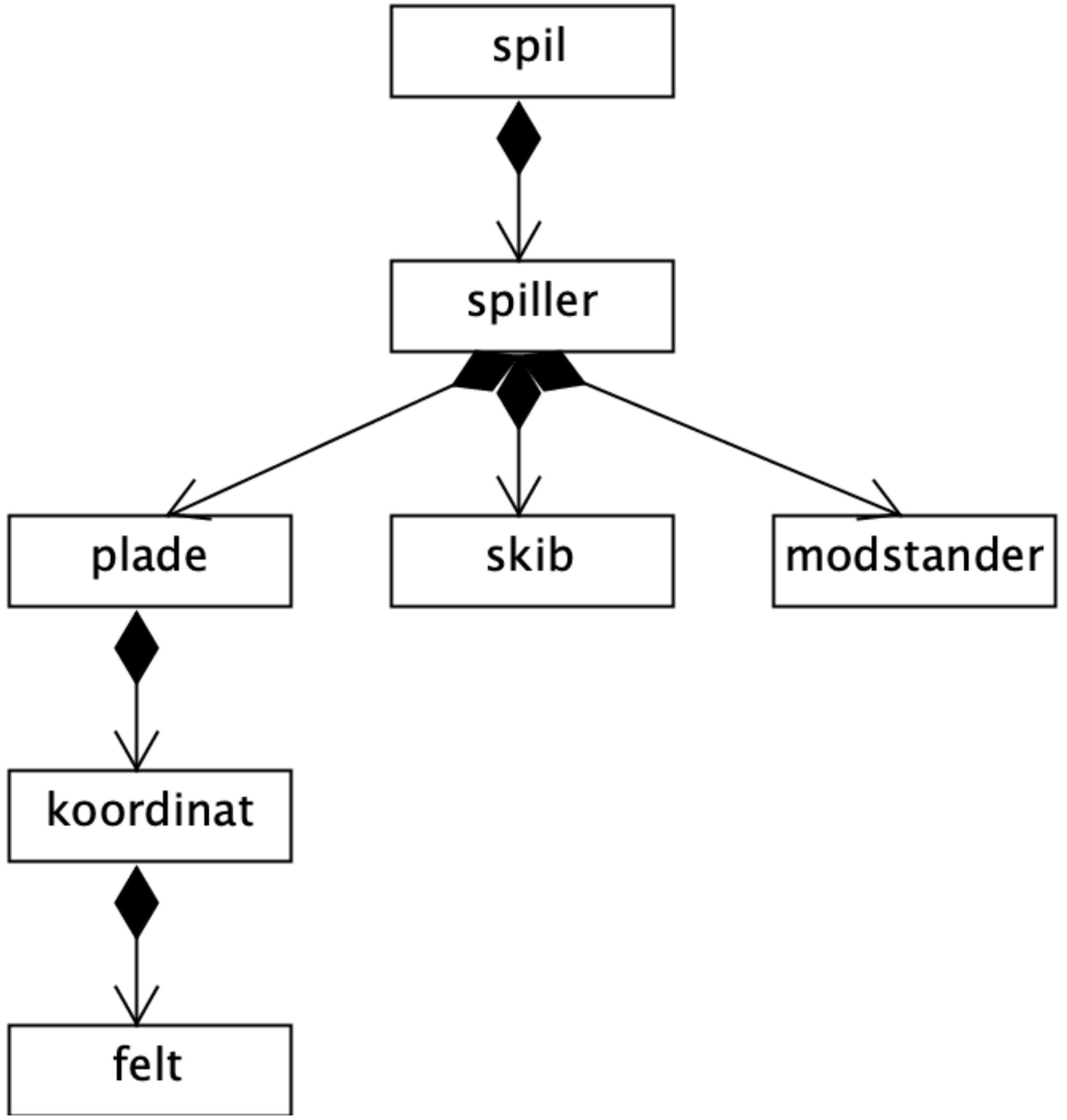
- Opgaven specificerer ikke, hvor mange skibe, der skal være i spillet, ej heller hvilken form de skal have.
- Typiske spil har 4-5 skibe, som er lige, ligger enten horisontalt eller verticalt, og har størrelse mellem 2 og 5 felter.
- Vi vælger 4 skibe, 2 små, 1 mellem og 1 stort. Vi sikrer at programmet nemt kan opdateres med et andet antal og størrelser.

```
/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> None)

/// A ship can be damaged
type ship(sz : int) =
  member this.size = sz

/// A player is a human player, which has 2 boards and
/// several ships
type player() =
  let my_board = board()
  let opponents_board = board()
  let mutable opponent = None
  let ships = List.map (fun elm -> ship(elm)) [2;2;3;5]

  member this.setOpponent(p : player) =
    opponent <- Some p

/// A game is a battleship game
type game() =
  let p1 = player()
  let p2 = player()
  do p1.setOpponent(p2)
  do p2.setOpponent(p1)
```

Sænke slagskibe: *Kender til relationer*

Problemer:

- Hvordan skal vi placere skibe på brættet?
- Hvordan skal vi repræsentere, at der er blevet skudt på et felt eller ej?
- Hvordan skal vi holde øje med om et skib er sænket eller ej?



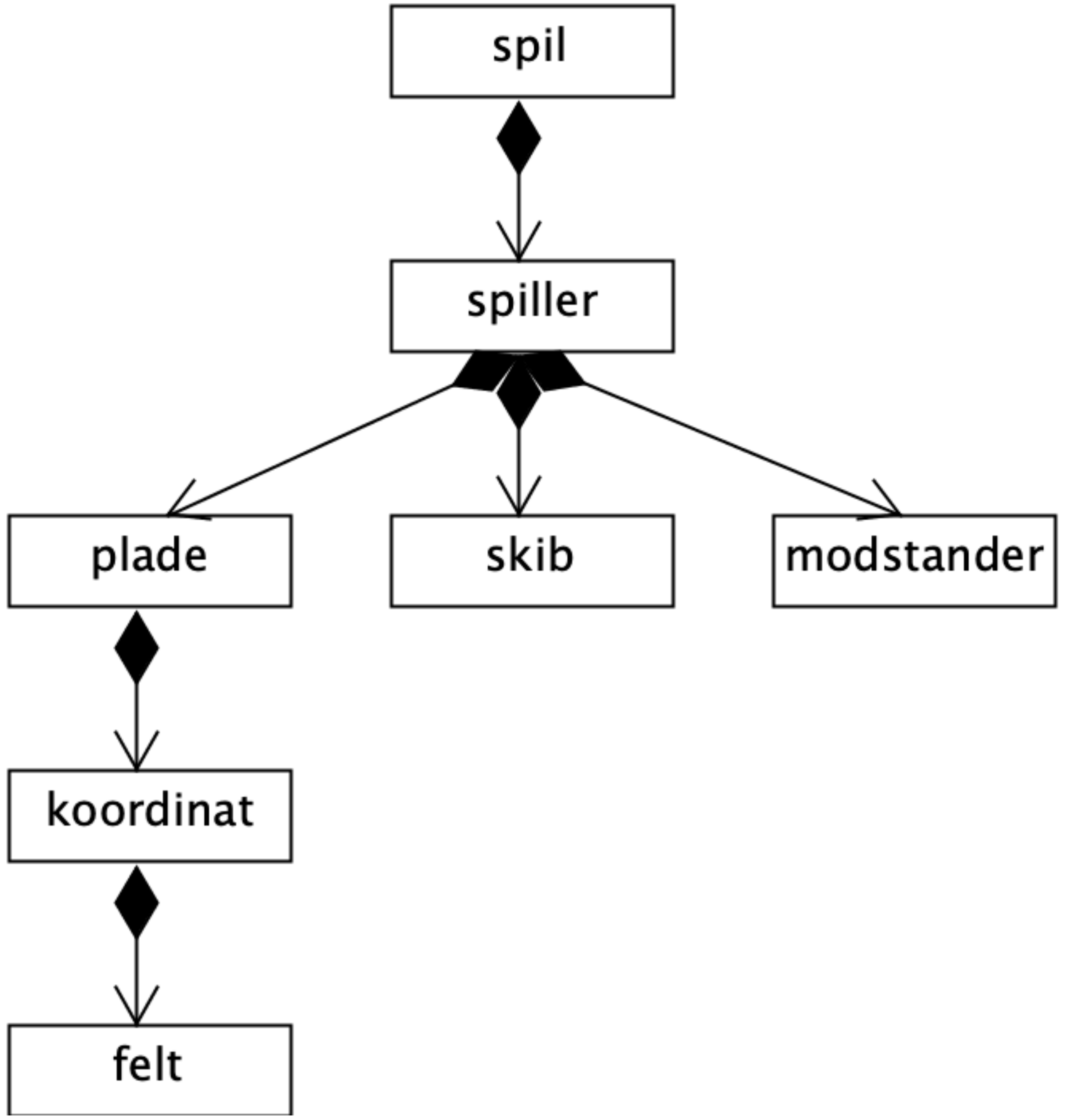
Sænke slagskibe: *Kender til relationer*

Problemer:

- Hvordan skal vi placere skibe på brættet?
- Hvordan skal vi repræsentere, at der er blevet skudt på et felt eller ej?
- Hvordan skal vi holde øje med om et skib er sænket eller ej?

Princip:

- Objekter skal være simple med så få bindinger til andre, som muligt

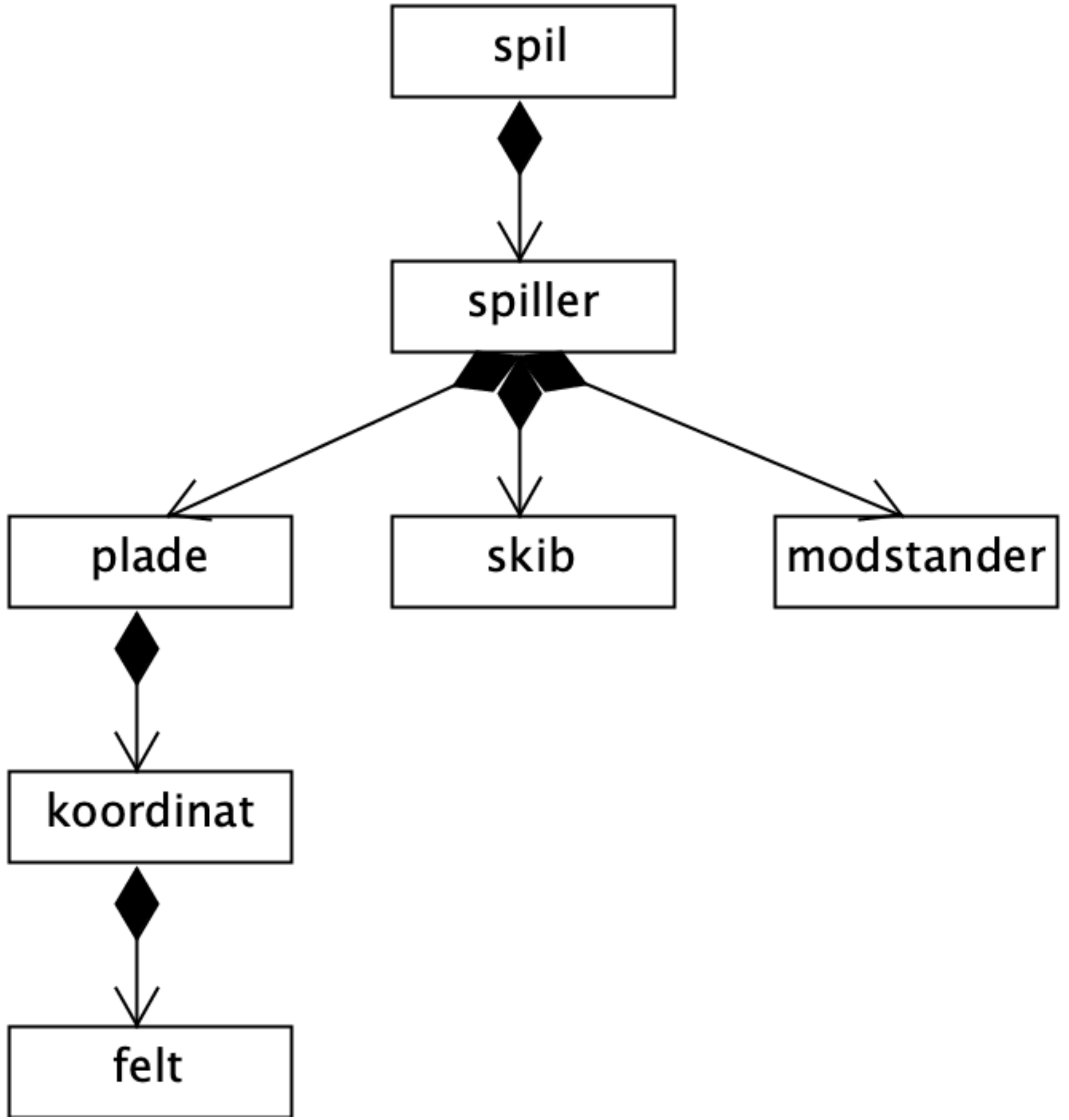


Sænke slagskibe: *Kender til relationer*

Problemer:
- Hvordan skal vi placere skibe på brættet?
- Hvordan skal vi repræsentere, at der er blevet skudt på et felt eller ej?
- Hvordan skal vi holde øje med om et skib er sænket eller ej?

Princip:
- Objekter skal være simple med så få bindinger til andre, som muligt

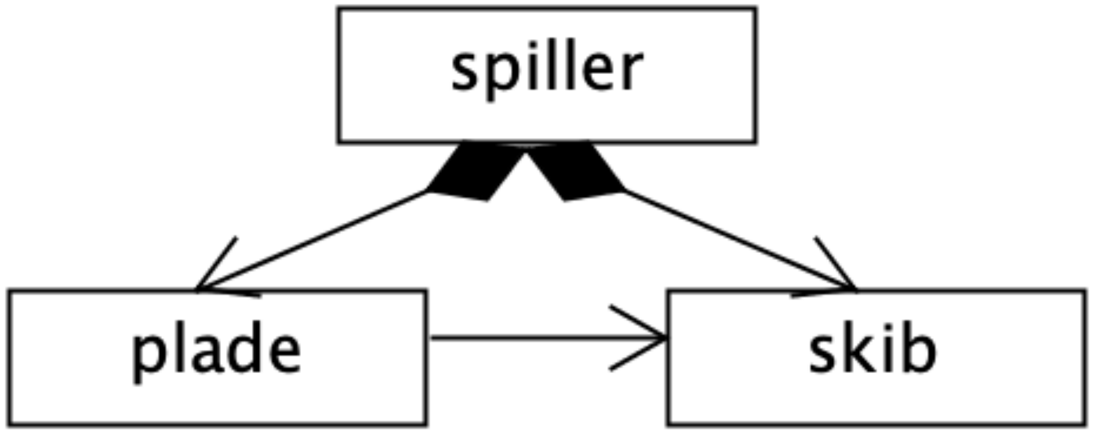
- Spiller har skibe
- Skibe kender til spiller/plade/felt/koordinat?
- Plade kender til spiller/skibe?



Sænke slagskibe: *Kender til relationer*

Problemer:

- Hvordan skal vi placere skibe på brættet?
- Hvordan skal vi repræsentere, at der er blevet skudt på et felt eller ej?
- Hvordan skal vi holde øje med om et skib er sænket eller ej?



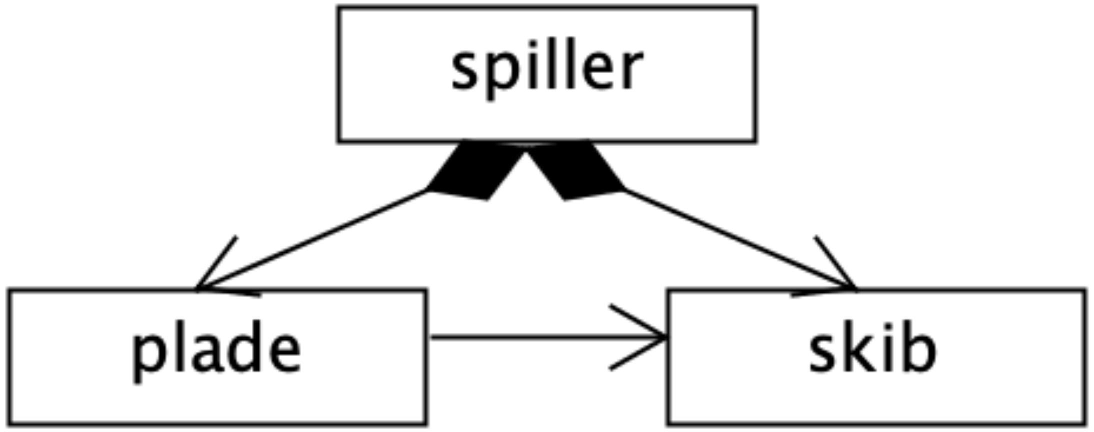
Princip:

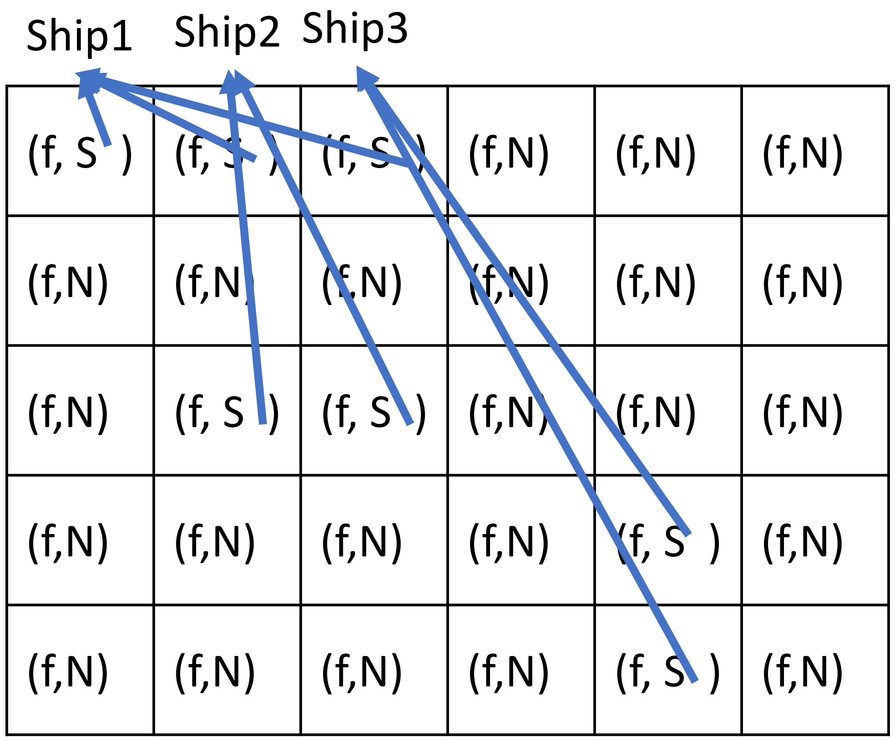
- Objekter skal være simple med så få bindinger til andre, som muligt

- Spiller har skibe
- Skibe kender til spiller/plade/felt/koordinat?
- Plade kender til spiller/skibe?

```
/// A field's state may have been shot at and may be occupied by a
/// ship
type State = bool * (ship option)

/// A board is a square set of fields with row-column
/// coordinates. Ships are placed on the board
type board() =
  let fields = Array2D.init 10 10 (fun i j -> (false, None) : State)
```

Sænke slagskibe: *Kender til relationer*

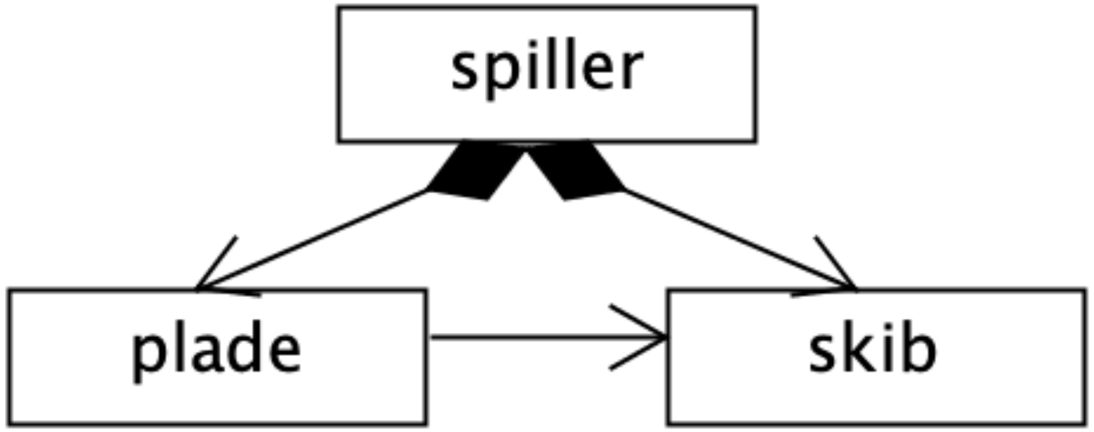


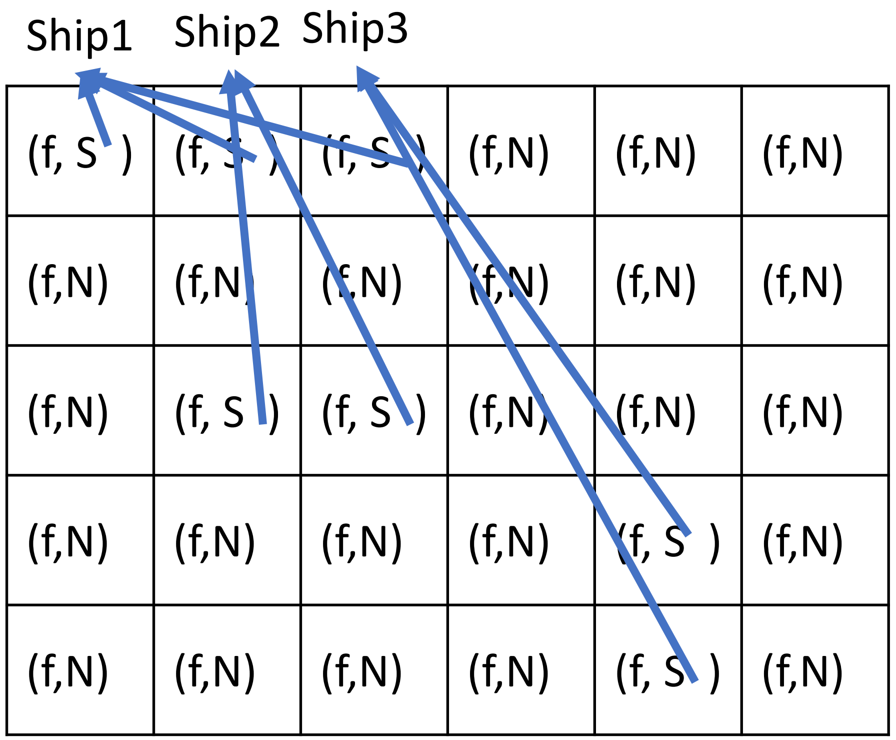


(f, S)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f,N)	(f,N)
(f,N)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)

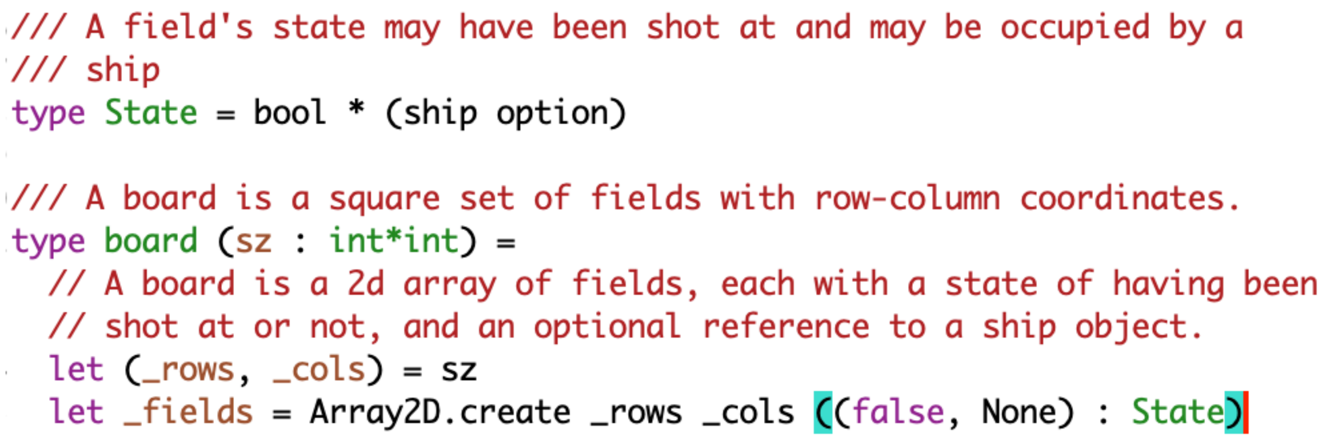
```
/// A field's state may have been shot at and may be occupied by a
/// ship
type State = bool * (ship option)

/// A board is a square set of fields with row-column coordinates.
type board (sz : int*int) =
  // A board is a 2d array of fields, each with a state of having been
  // shot at or not, and an optional reference to a ship object.
  let (_rows, _cols) = sz
  let _fields = Array2D.create _rows _cols ((false, None) : State)
```

Sænke slagskibe: *Kender til relationer*





(f, S)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f,N)	(f,N)
(f,N)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)



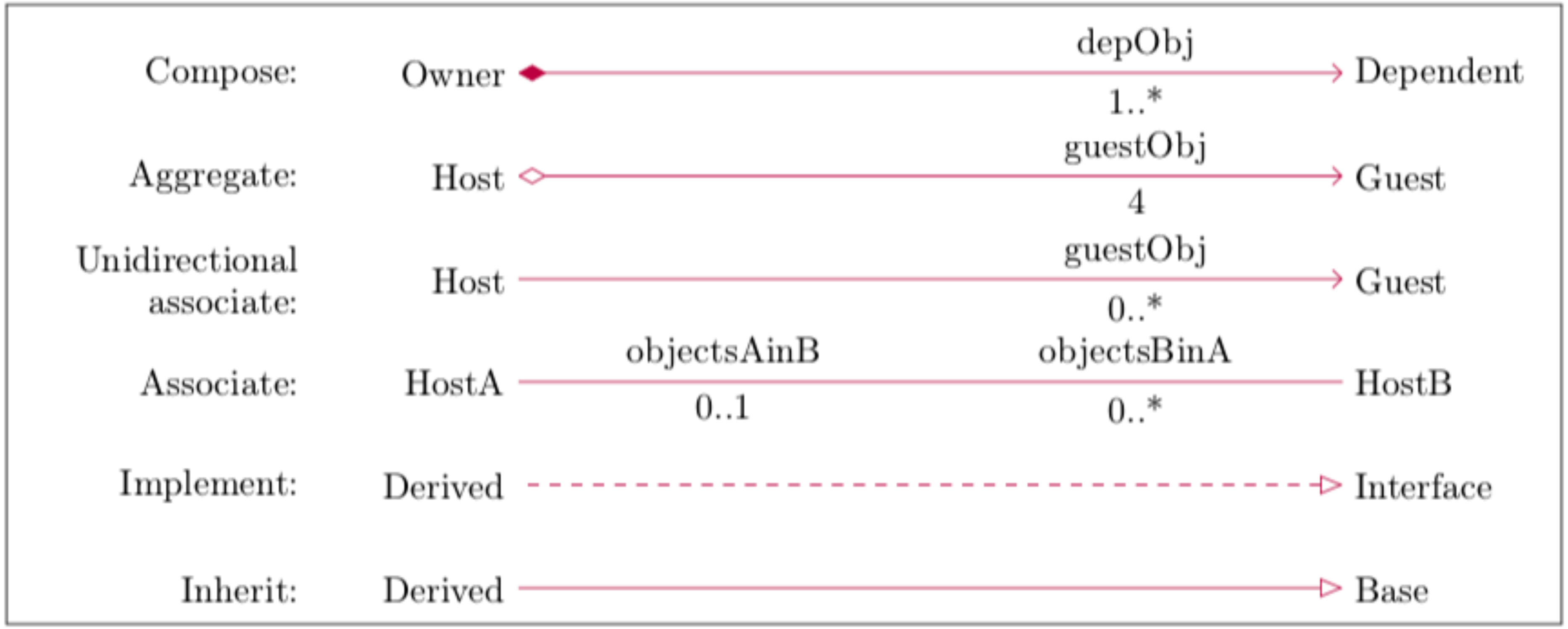
```
/// A field's state may have been shot at and may be occupied by a
/// ship
type State = bool * (ship option)

/// A board is a square set of fields with row-column coordinates.
type board (sz : int*int) =
  // A board is a 2d array of fields, each with a state of having been
  // shot at or not, and an optional reference to a ship object.
  let (_rows, _cols) = sz
  let _fields = Array2D.create _rows _cols ((false, None) : State)
```

Plade kender til skibe, skibe kender *ikke* til plade

Intermezzo: Relationer

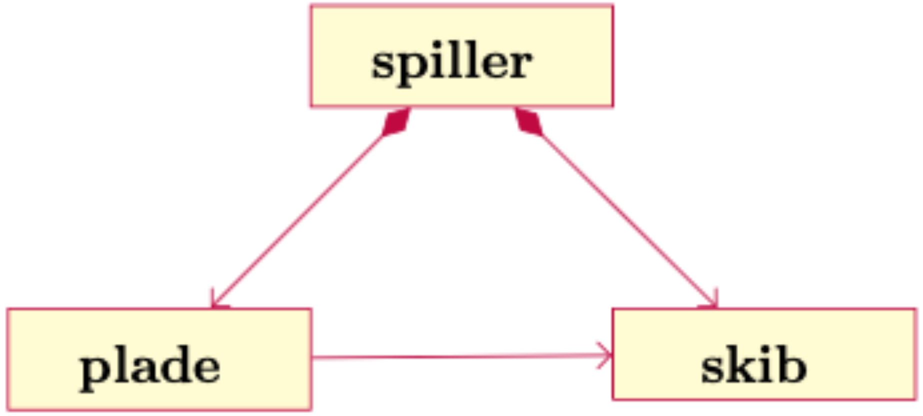
- Nedarvning: ‘is-a’
- Association: ‘has-a’
  - Association - Host ’kender til’ gæst
  - Aggregation - Host ’har kopi af’ gæst
  - Composition - Host ’opretter afhængig og har kopi af’ gæst. Når host slettes slettes gæst ligeså.

Relationsgrafik



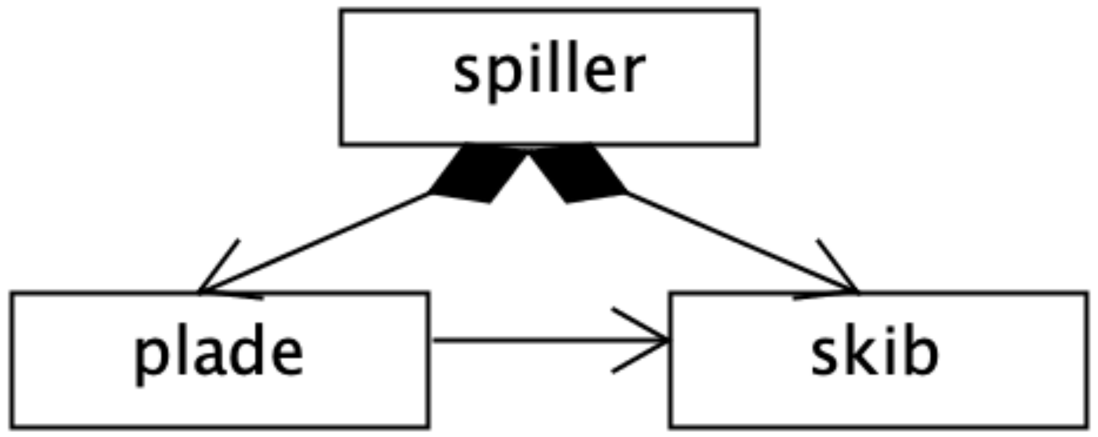
UML i LaTeX

```
\documentclass{article}
\usepackage[school,simplified]{pgf-umlcd}

\begin{document}
\begin{tikzpicture}
  \begin{class}[text width=5em, text height=1em]{spiller}{0,0}
  \end{class}
  \begin{class}[text width=5em, text height=1em]{plade}{-2,-2}
   \end{class}
   \begin{class}[text width=5em, text height=1em]{skib}{2,-2}
   \end{class}
  \composition{spiller}{}{}{plade}
  \unidirectionalAssociation{plade}{}{}{skib}
  \composition{spiller}{}{}{skib}
\end{tikzpicture}
\end{document}
```

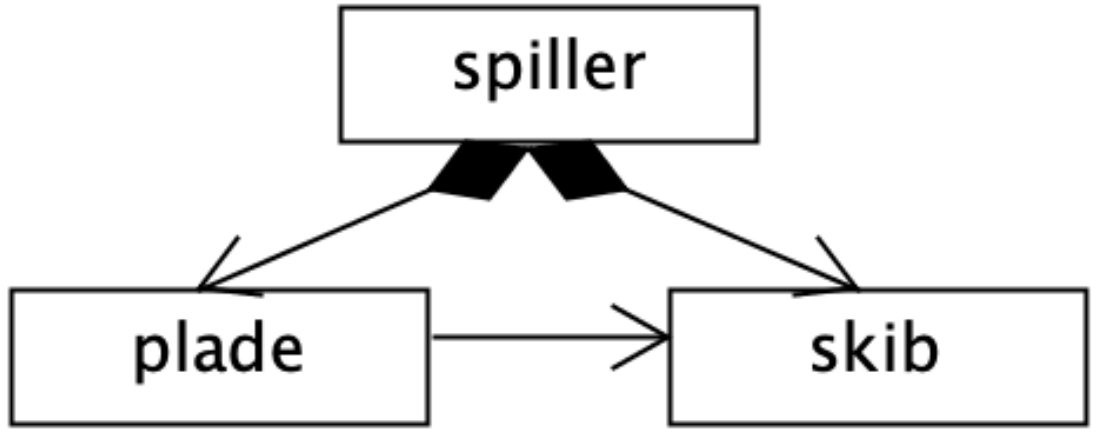


http://mirrors.dotsrc.org/ctan/graphics/pgf/contrib/pgf-umlcd/pgf-umlcd-manual.pdf

Princip: simple klasser nederst



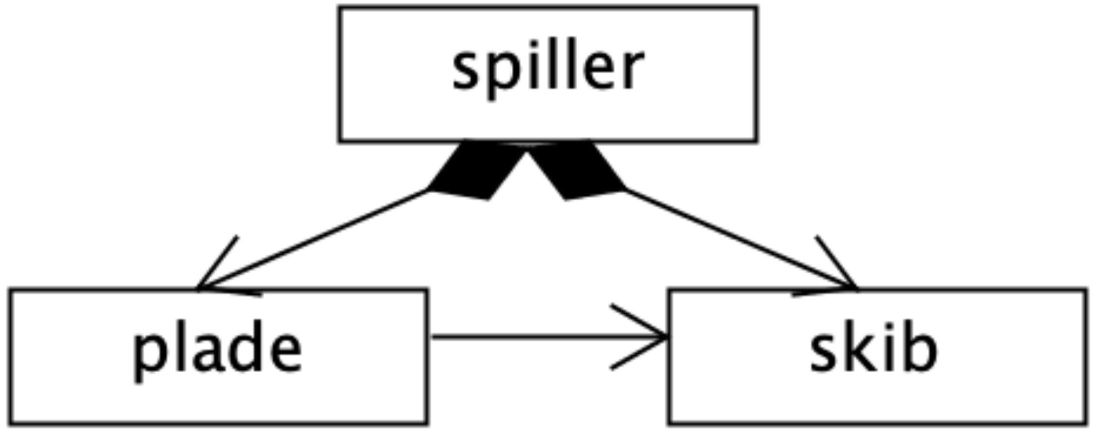
Princip: simple klasser nederst

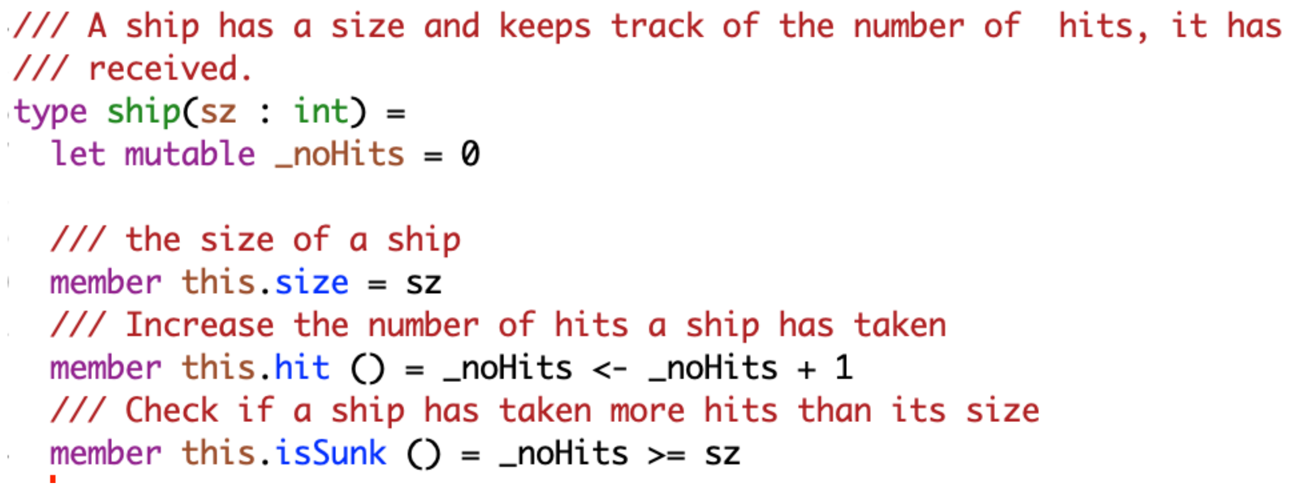


Ide:

- Skibe har en størrelse,
- de kan tage en antal hits,
- når antallet af hits er større end skibets størrelse, så er skibet sunket.

Princip: simple klasser nederst



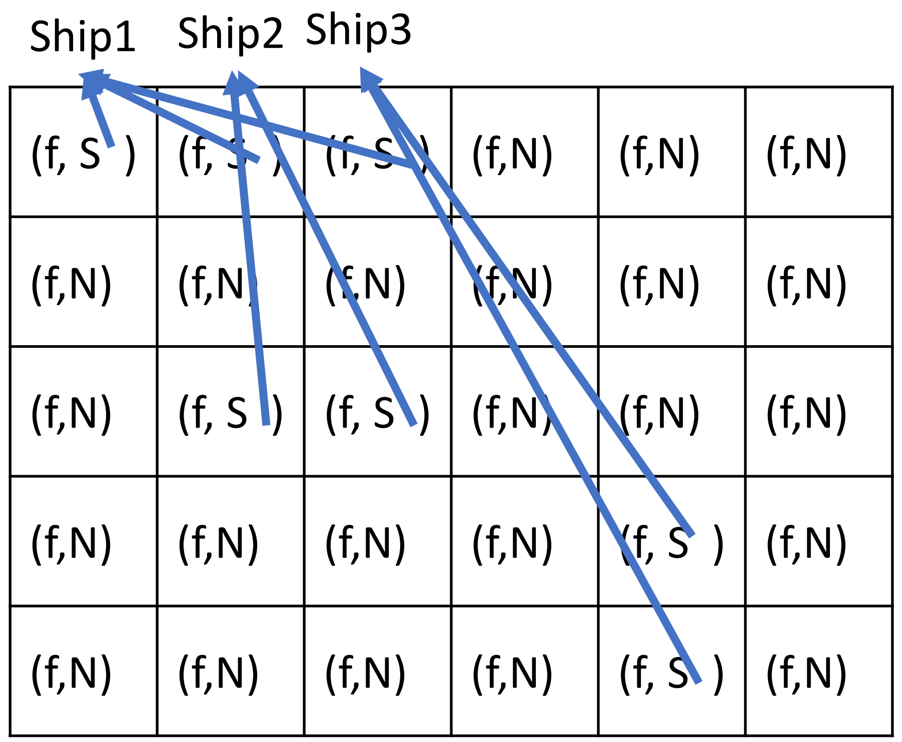


```
/// A ship has a size and keeps track of the number of  hits, it has
/// received.
type ship(sz : int) =
  let mutable _noHits = 0

  /// the size of a ship
  member this.size = sz
  /// Increase the number of hits a ship has taken
  member this.hit () = _noHits <- _noHits + 1
  /// Check if a ship has taken more hits than its size
  member this.isSunk () = _noHits >= sz
```

Ide:

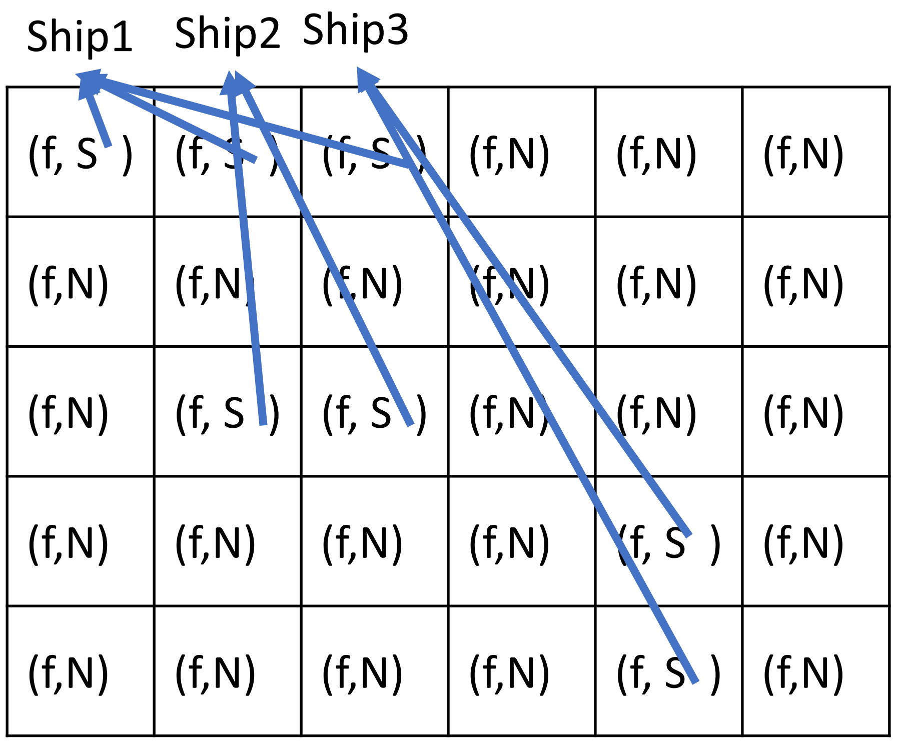
- Skibe har en størrelse,
- de kan tage en antal hits,
- når antallet af hits er større end skibets størrelse, så er skibet sunket.

Sænke slagskibe: *ToString()*

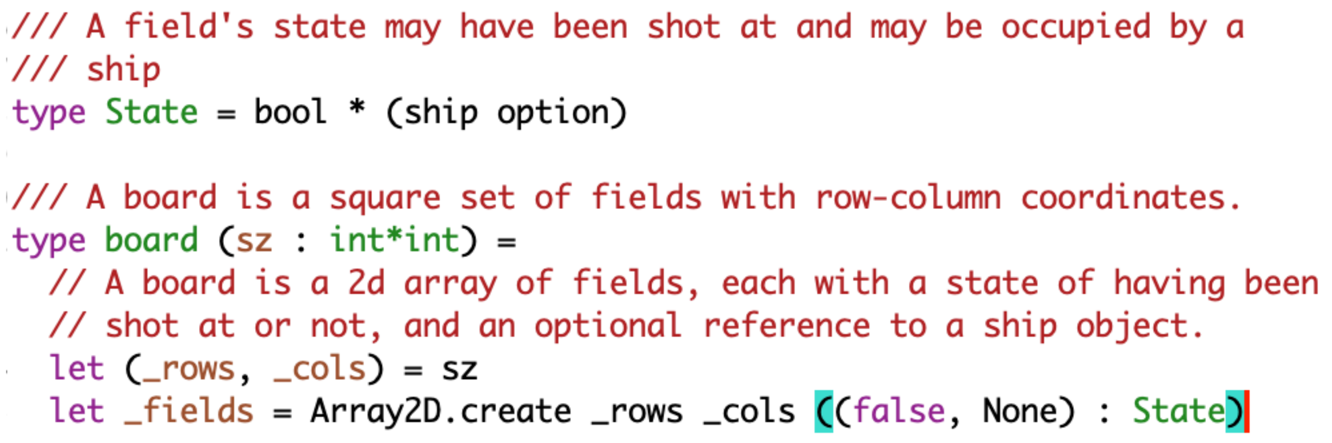

Ship1
Ship2
Ship3
(f, S) (f, S) (f, S) (f,N) (f,N) (f,N)
(f,N) (f,N) (f,N) (f,N) (f,N) (f,N)
(f,N) (f, S) (f, S) (f,N) (f,N) (f,N)
(f,N) (f,N) (f,N) (f,N) (f, S) (f,N)
(f,N) (f,N) (f,N) (f,N) (f, S) (f,N)

```
/// A field's state may have been shot at and may be occupied by a
/// ship
type State = bool * (ship option)

/// A board is a square set of fields with row-column coordinates.
type board (sz : int*int) =
  // A board is a 2d array of fields, each with a state of having been
  // shot at or not, and an optional reference to a ship object.
  let (_rows, _cols) = sz
  let _fields = Array2D.create _rows _cols ((false, None) : State)
```

Sænke slagskibe: *ToString()*



(f, S)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f,N)	(f,N)
(f,N)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)

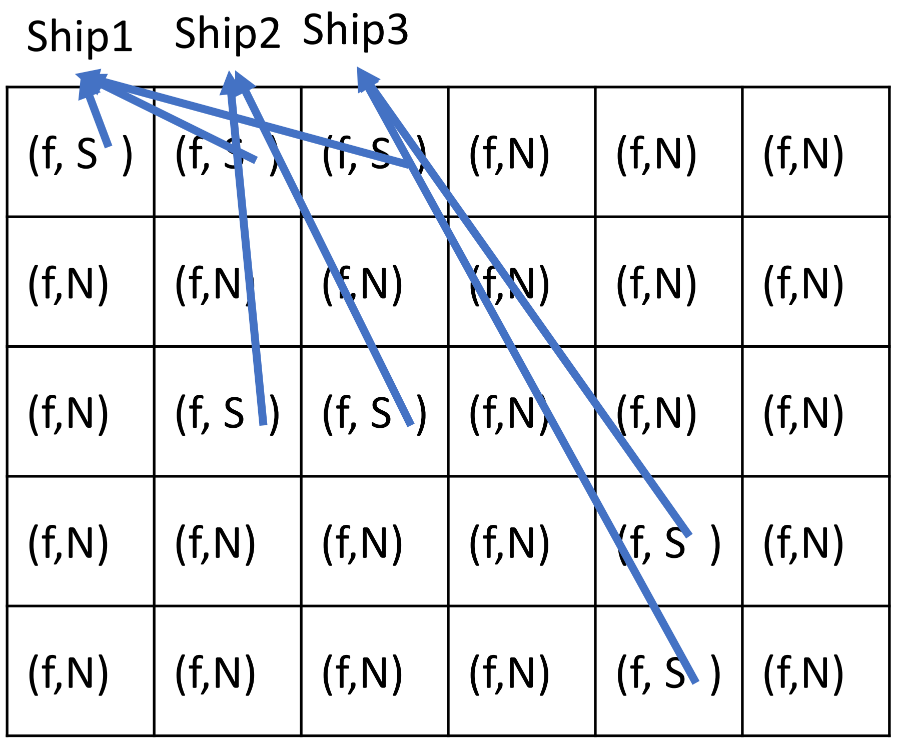


```
/// A field's state may have been shot at and may be occupied by a
/// ship
type State = bool * (ship option)

/// A board is a square set of fields with row-column coordinates.
type board (sz : int*int) =
  // A board is a 2d array of fields, each with a state of having been
  // shot at or not, and an optional reference to a ship object.
  let (_rows, _cols) = sz
  let _fields = Array2D.create _rows _cols ((false, None) : State)
```

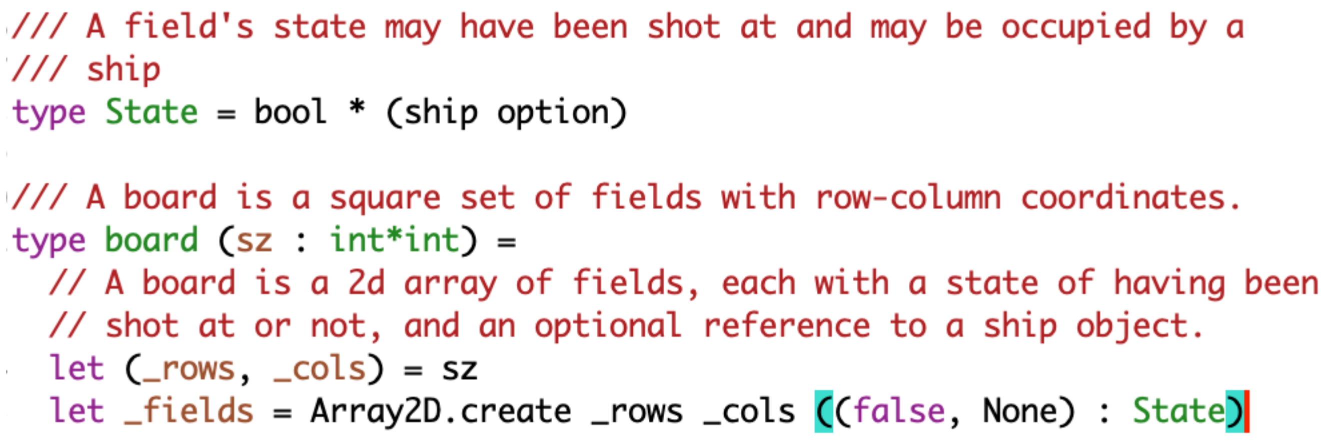
Understøttelse af printf:

```
let b = board()
printfn “%A” b
```

Sænke slagskibe: *ToString()*



(f, S)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f,N)	(f,N)
(f,N)	(f, S)	(f, S)	(f,N)	(f,N)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)
(f,N)	(f,N)	(f,N)	(f,N)	(f, S)	(f,N)



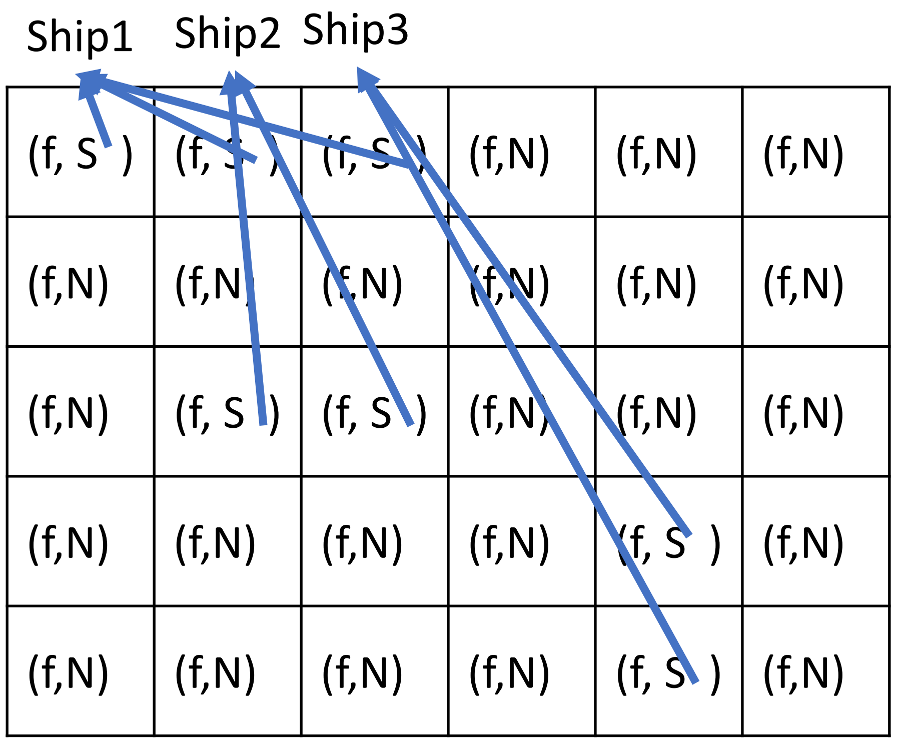
```
/// A field's state may have been shot at and may be occupied by a
/// ship
type State = bool * (ship option)

/// A board is a square set of fields with row-column coordinates.
type board (sz : int*int) =
  // A board is a 2d array of fields, each with a state of having been
  // shot at or not, and an optional reference to a ship object.
  let (_rows, _cols) = sz
  let _fields = Array2D.create _rows _cols ((false, None) : State)
```

Understøttelse af printf:

```
let b = board()
printfn “%A” b
```

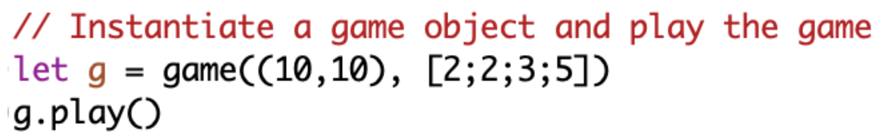
- Alle klasser nedarver fra System.Object klassen.
- System.Object har member: “ToString()”.
- Nedarvede members kan overskrives med “override”

Sænke slagskibe: *ToString()*


Ship1
Ship2
Ship3
(f, S) (f, S) (f, S) (f,N) (f,N) (f,N)
(f,N) (f,N) (f,N) (f,N) (f,N) (f,N)
(f,N) (f, S) (f, S) (f,N) (f,N) (f,N)
(f,N) (f,N) (f,N) (f,N) (f, S) (f,N)
(f,N) (f,N) (f,N) (f,N) (f, S) (f,N)

```
type board (sz : int*int) =
  // A board is a 2d array of fields, each with a state of having been
  // shot at or not, and an optional reference to a ship object.
  let (_rows, _cols) = sz
  let _fields = Array2D.create _rows _cols ((false, None) : State)

  /// Produce a string-representation of the board as a table with the
  /// sympols " " - no shot taken and no ship, "s" - no shot take and
  /// a ship present, "m" - shot take but not ship present, "h" shot
  /// take and ship hit.
  override this.ToString () =
    let digits = max _rows _cols |> float |> log10 |> ceil |> int
    let mutable str = (String.replicate (digits+1) " ")
    for j = 1 to _cols do
      str <- str + (sprintf "%*d " digits j)
    str <- str + "\n"
    for i = 1 to _rows do
      str <- str + (sprintf "%*d " digits i)
      for j = 1 to _cols do
        let sym =
          match _fields.[i-1,j-1] with
            (true, None ) -> "m"
            | (true, Some s) -> "h"
            | (false, Some s) -> "s"
            | _ -> " "
        str <- str + (sprintf "%*s " digits sym)
      str <- str + "\n"
    str.[..str.Length-2]

let b = board (10,10)
printfn "My board:\n%A" b
```

Metode: programmér bagfra



```
// Instantiate a game object and play the game
let g = game((10,10), [2;2;3;5])
g.play()
```

Konsekvens: spiller behøver ikke at have association til modstander

Metode: programmér bagfra

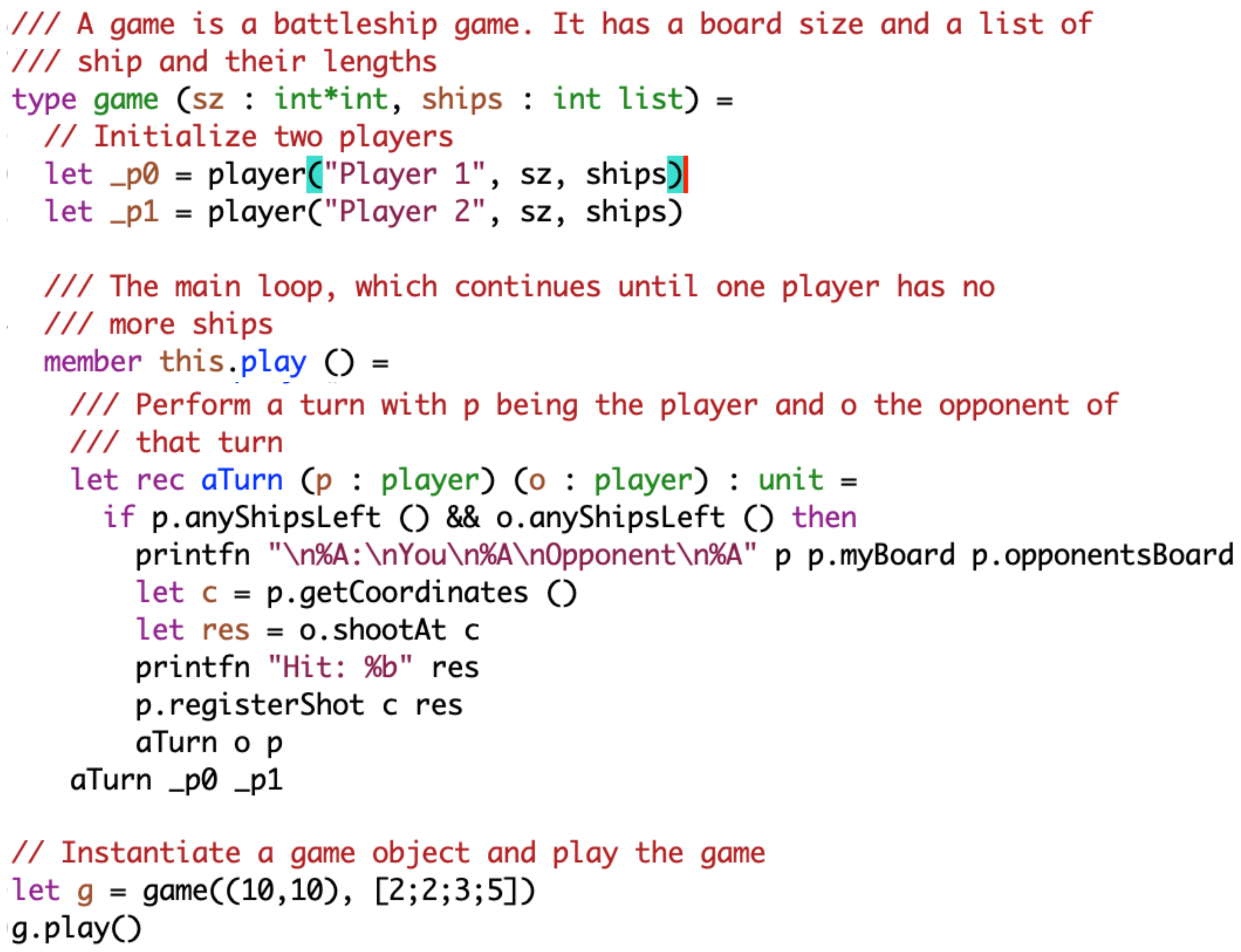
```
/// A game is a battleship game. It has a board size and a list of
/// ship and their lengths
type game (sz : int*int, ships : int list) =
  // Initialize two players
  let _p0 = player("Player 1", sz, ships)
  let _p1 = player("Player 2", sz, ships)

  /// The main loop, which continues until one player has no
  /// more ships
  member this.play () =
```

Rekursion eller while-løkke?

```
// Instantiate a game object and play the game
let g = game((10,10), [2;2;3;5])
g.play()
```

Konsekvens: spiller behøver ikke at have association til modstander

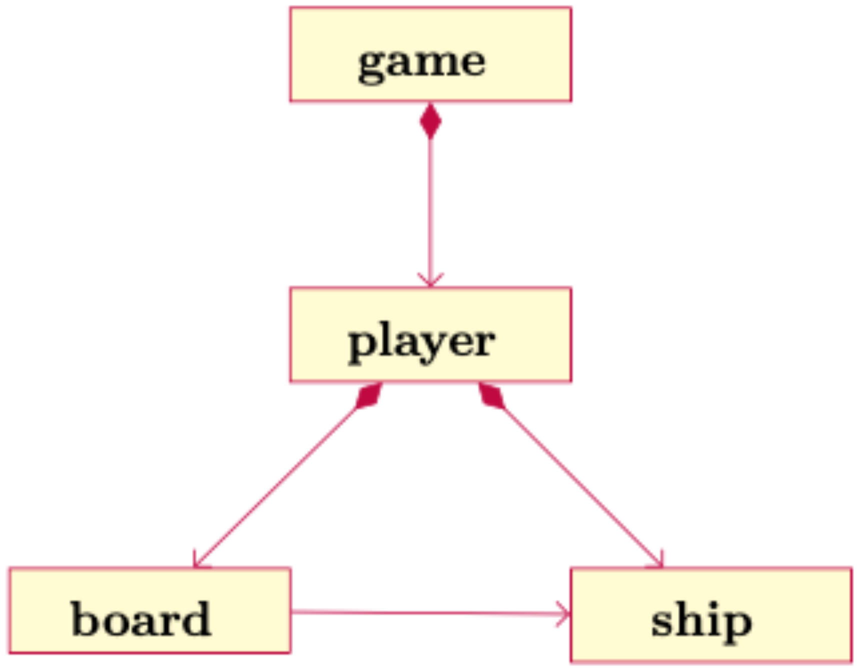
Metode: programmér bagfra

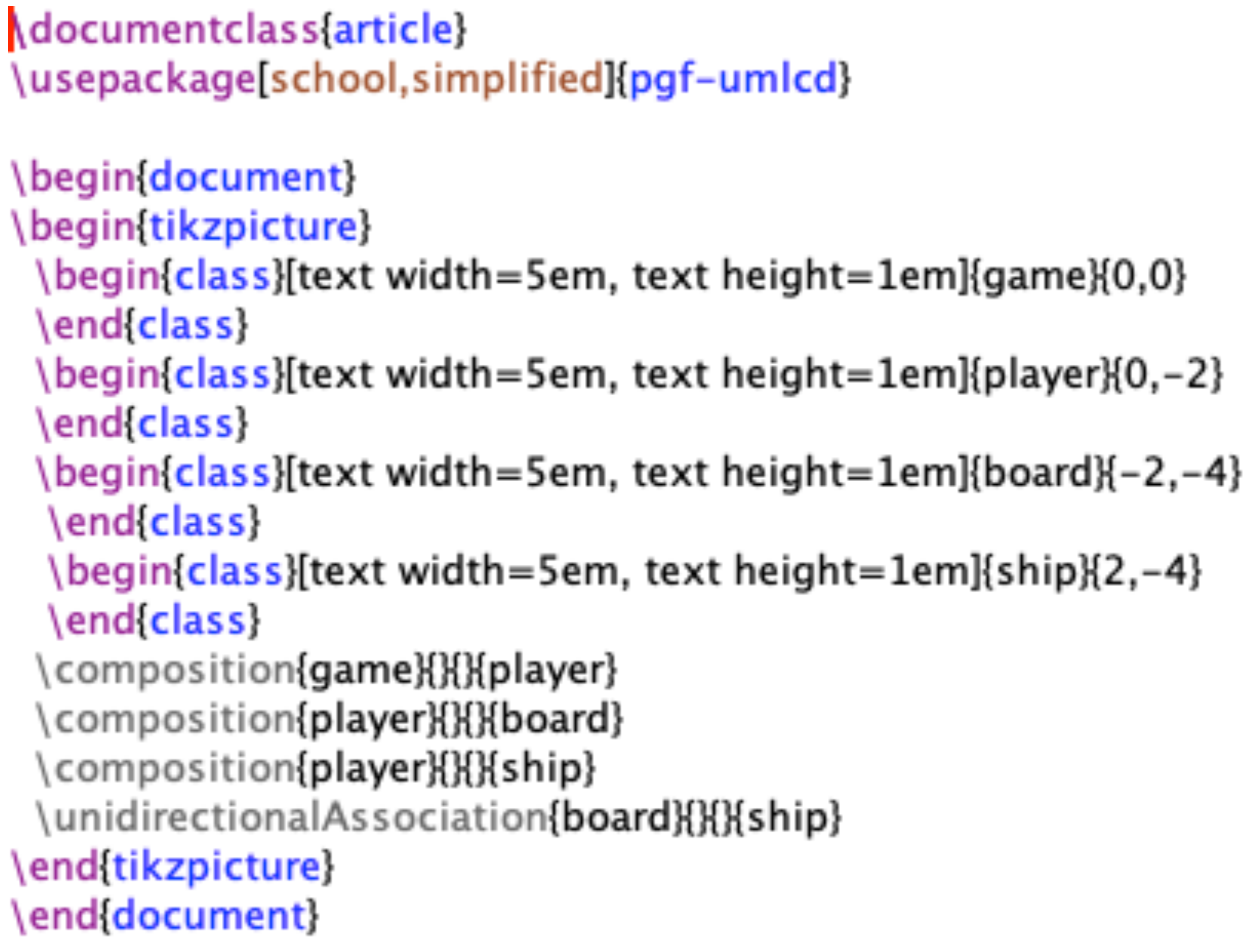


```
/// A game is a battleship game. It has a board size and a list of
/// ship and their lengths
type game (sz : int*int, ships : int list) =
  // Initialize two players
  let _p0 = player("Player 1", sz, ships)
  let _p1 = player("Player 2", sz, ships)

  /// The main loop, which continues until one player has no
  /// more ships
  member this.play () =
    /// Perform a turn with p being the player and o the opponent of
    /// that turn
    let rec aTurn (p : player) (o : player) : unit =
      if p.anyShipsLeft () && o.anyShipsLeft () then
        printfn "\n%A:\nYou\n%A\nOpponent\n%A" p p.myBoard p.opponentsBoard
        let c = p.getCoordinates ()
        let res = o.shootAt c
        printfn "Hit: %b" res
        p.registerShot c res
        aTurn o p
    aTurn _p0 _p1

// Instantiate a game object and play the game
let g = game((10,10), [2;2;3;5])
g.play()
```

Konsekvens: spiller behøver ikke at have association til modstander

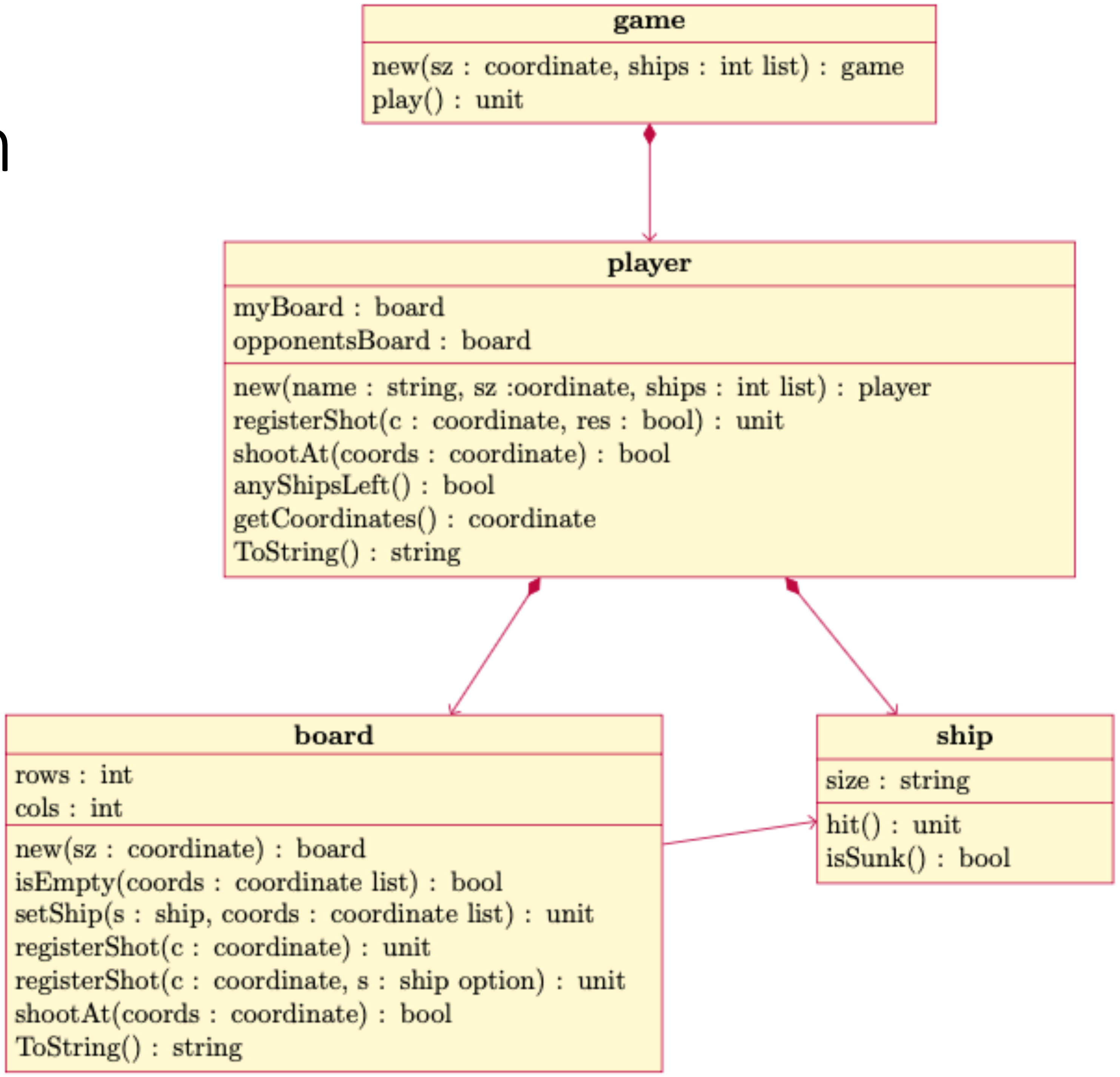
Final design





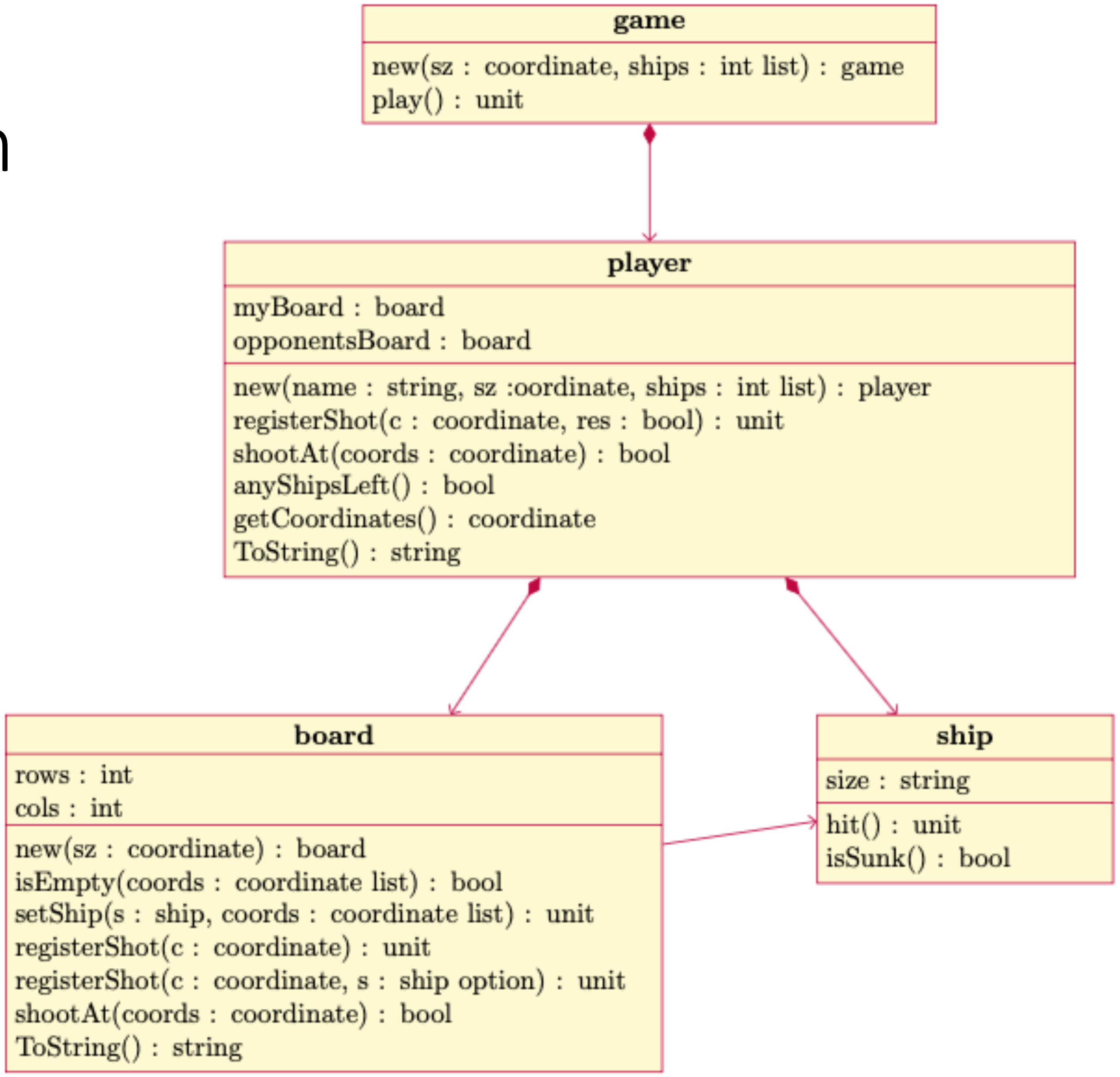
```
\documentclass{article}
\usepackage[school,simplified]{pgf-umlcd}

\begin{document}
\begin{tikzpicture}
  \begin{class}[text width=5em, text height=1em]{game}{0,0}
  \end{class}
  \begin{class}[text width=5em, text height=1em]{player}{0,-2}
  \end{class}
  \begin{class}[text width=5em, text height=1em]{board}{-2,-4}
   \end{class}
   \begin{class}[text width=5em, text height=1em]{ship}{2,-4}
   \end{class}
  \composition{game}{}{}{player}
  \composition{player}{}{}{board}
  \composition{player}{}{}{ship}
  \unidirectionalAssociation{board}{}{}{ship}
\end{tikzpicture}
\end{document}
```

Final design, with properites and methods


game
new(sz : coordinate, ships : int list) : game
play() : unit
player
myBoard : board
opponentsBoard : board
new(name : string, sz :oordinate, ships : int list) : player
registerShot(c : coordinate, res : bool) : unit
shootAt(coords : coordinate) : bool
anyShipsLeft() : bool
getCoordinates() : coordinate
ToString() : string
board
rows : int
cols : int
new(sz : coordinate) : board
isEmpty(coords : coordinate list) : bool
setShip(s : ship, coords : coordinate list) : unit
registerShot(c : coordinate) : unit
registerShot(c : coordinate, s : ship option) : unit
shootAt(coords : coordinate) : bool
ToString() : string
ship
size : string
hit() : unit
isSunk() : bool

Final design, with properites and methods

```
battleship.tex
battleship.fsx
```


game
new(sz : coordinate, ships : int list) : game
play() : unit
player
myBoard : board
opponentsBoard : board
new(name : string, sz :oordinate, ships : int list) : player
registerShot(c : coordinate, res : bool) : unit
shootAt(coords : coordinate) : bool
anyShipsLeft() : bool
getCoordinates() : coordinate
ToString() : string
board
rows : int
cols : int
new(sz : coordinate) : board
isEmpty(coords : coordinate list) : bool
setShip(s : ship, coords : coordinate list) : unit
registerShot(c : coordinate) : unit
registerShot(c : coordinate, s : ship option) : unit
shootAt(coords : coordinate) : bool
ToString() : string
ship
size : string
hit() : unit
isSunk() : bool

```
battleship.tex
battleship.fsx
```

Function overloading

Problem:
Modstanders plade kendes kun delvist, men ToString() afhænger af viden om der er et skib eller ej.

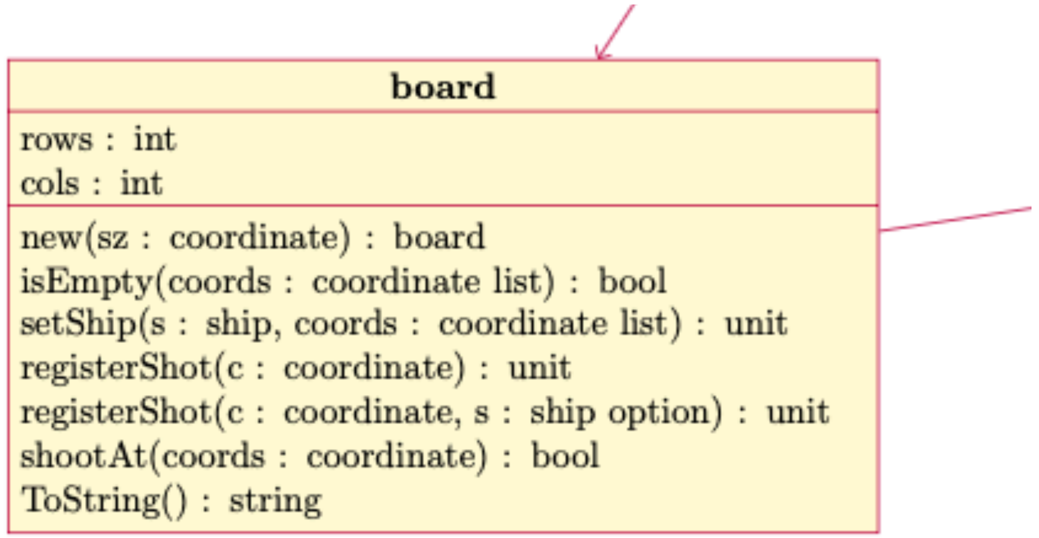
Løsning:
Opret et ‘dummy’ skib, og indsæt ved hit på spillers kopi af modstanders bræt.

Function overloading

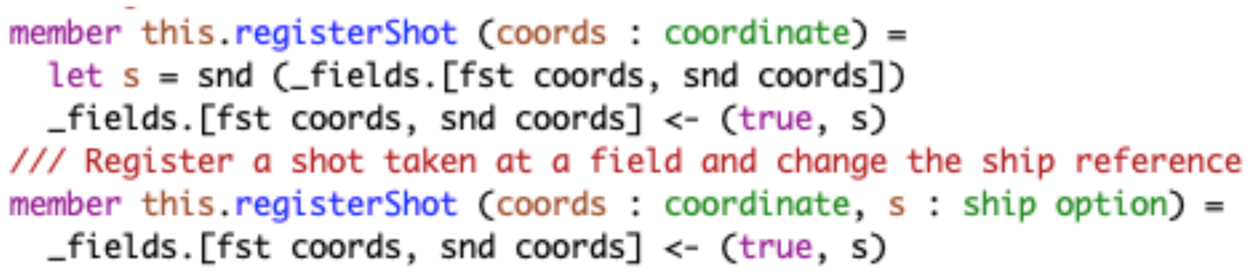
battleship.tex
battleship.fsx

Problem:
Modstanders plade kendes kun delvist, men ToString() afhænger af viden om der er et skib eller ej.

Løsning:
Opret et ‘dummy’ skib, og indsæt ved hit på spillers kopi af modstanders bræt.



```
board
rows : int
cols : int
new(sz : coordinate) : board
isEmpty(coords : coordinate list) : bool
setShip(s : ship, coords : coordinate list) : unit
registerShot(c : coordinate) : unit
registerShot(c : coordinate, s : ship option) : unit
shootAt(coords : coordinate) : bool
ToString() : string
```



```
member this.registerShot (coords : coordinate) =
  let s = snd (_fields.[fst coords, snd coords])
  _fields.[fst coords, snd coords] <- (true, s)
/// Register a shot taken at a field and change the ship reference
member this.registerShot (coords : coordinate, s : ship option) =
  _fields.[fst coords, snd coords] <- (true, s)
```