Vidéo

Vidéo

Stratégies et problèmes

repairOtherShip: La stratégie que j'utilise dans cette fonction est de me concentrer sur le PatrolBoat. Puisque que le PatrolBoat est celui qui a le pouvoir de réparer les autres bateaux, je trouve qu'il est important qu'il reste en vie le plus longtemps possible. Ensuite, si le PatrolBoat est pleinement réparé, je vérifie la vie restante de tous les autres bateaux, et je répare le bateau qui est le plus faible/détruit. Il pourrait être plus optimisé en choisissant l'ordre d'importance de chaque bateau, mais je trouve que l'essentielle est le PatrolBoat.

```
let repairShip (name: Name) (grid: Sector Grid) =
    let (hitSectors, _) = Battlefield.getHitSectorsOfShip name grid
    if (List.length hitSectors) = 0 then
        grid
    else
        let (name, pos, _, x, y) = List.head hitSectors
        let repairedSector = Active (name, pos, false)
        Battlefield.setSector grid (x,y) repairedSector

let getShipToRepair shipWithHealth =
    match shipWithHealth with
    | (name, _) -> name

let repairOtherShip (grid: Sector Grid) : Sector Grid =
    let patrolBoatHealth = Battlefield.getRemainingHealth PatrolBoat grid
    let destroyerHealth = Battlefield.getRemainingHealth Destroyer grid
    let submarineHealth = Battlefield.getRemainingHealth Submarine grid
    let cruiserHealth = Battlefield.getRemainingHealth Cruiser grid
    let aircraftCarrierHealth = Battlefield.getRemainingHealth AircraftCarrier grid

    let allHealth = List.sortBy (fun (_, health) -> health) [(PatrolBoat, patrolBoatHealth); (Destroyer, destroyerHealth); (Submarine, submarineHealth); (Cruiser, cruiserHealth); (AircraftCarrier, aircraftCarrierHealth)]
    let allHealth = List.filter (fun (_, health) -> health > 0) allHealth //Remove sunk ships
    let allHealth = List.filter (fun (name, health) ->
        let (_, maxHealth) = Battlefield.getHitSectorsOfShip name grid
        health <> maxHealth) allHealth //Remove full health ships

    //Focus on repairing patrol boat first
    if patrolBoatHealth > 0 && patrolBoatHealth <> 2 then
        repairShip PatrolBoat grid
    //If patrol boat is full health, repair lowest health ship
    elif List.length allHealth > 0 then
        repairShip (getShipToRepair(List.head allHealth)) grid
    else
        grid
```

repairSpy : Je trouve qu'il n'y a pas grande intelligence à faire dans cette fonction. Je fais simplement vérifier si un est secteur du Spy est détruit, et si oui je le répare.

```
let repairSpy (grid: Sector Grid) : Sector Grid =
    repairShip Spy grid
```

launchTorpedo : Pour cette fonction, je calcule la coordonnée du périmètre du bateau Submarine la plus près d'une des coordonnées du Spy. Je place ensuite une torpille à cette coordonnée.

```fsharp
//Calculate the closest coord/sector
let calculateClosest spyCoords subMarinePerim =

    let calculateDist (x1,y1) (x2,y2) =
        abs(x1 - x2) + abs(y1 - y2)

    let rec compare spyCoord subPerim closestCoord minDist =
        match subPerim with
        | [] -> (closestCoord, minDist)
        | coord::rest ->
            let dist = calculateDist spyCoord coord
            if dist < minDist then
                let closestCoord' = coord
                let minDist' = dist
                compare spyCoord rest closestCoord' minDist'
            else
                compare spyCoord rest closestCoord minDist

    let rec findClosestCoord coords closestCoord minDist =
        match coords with
        | [] -> (closestCoord, minDist)
        | coord::rest ->
            let (closestCoord', minDist') = compare coord subMarinePerim closestCoord minDist
            findClosestCoord rest closestCoord' minDist'

    let (finalCoord, finalDist) = findClosestCoord spyCoords (0,0) System.Int32.MaxValue
    finalCoord


let launchTorpedo (grid: Sector Grid) : Sector Grid =
    let dims = Navigation.getDimsGrid grid
    let submarineCoords = getShipCoords Submarine grid
    let submarinePerimeter = getPerimeterCoords submarineCoords dims //Get a list of perimeter coords of the submarine

    let spyCoords = getShipCoords Spy grid  //Get a list of coord of spy

    let closestCoord = calculateClosest spyCoords submarinePerimeter    //Find the closest coord of submarine perim coord to spy coords

    Battlefield.setSector grid closestCoord Torpedo
```

advanceTorpedo : La stratégie que j'utilise dans cette fonction est similaire à la stratégie que j'utilise dans **launchTorpedo**. En principe, je prends les positions cardinales autour de chaque Torpille, et je calcule la position la plus près du Spy. Si la position est la coordonnée d'un autre bateau, j'enlève cette position de la liste des positions de mouvement possible, de sorte que la torpille n'atterrit pas sur un bateau allié à celle-ci.

```
let getNewTorpedoPositions (x,y) grid =
    let (dimx, dimy) = Navigation.getDimsGrid grid
    let cardinalPositions =
        [
        (x + 1, y);
        (x - 1, y);
        (x, y + 1);
        (x, y - 1);
        ]
    let allSectors = Grid.getAllSector grid getAllActiveSectors  //Get a list of all active sectors
    let allShipSectors = List.filter (fun (name, _, _, _, _) -> name <> Spy) allSectors //Get a list of ship sectors except Spy
    let allShipCoords = List.map (fun (_, _, _, x, y) -> (x,y)) allShipSectors //Get a list of ship coords

    let newTorpedoPositions = List.filter (fun (x,y) -> x >= 0 && x < dimx && y >= 0 && y < dimy) cardinalPositions
    let newTorpedoPositions = List.filter (fun (x,y) -> not (List.contains (x,y) allShipCoords)) newTorpedoPositions
    newTorpedoPositions

let advanceTorpedoes (grid: Sector Grid) : Sector Grid =
    let spyCoords = getShipCoords Spy grid  //Get a list of coord of spy
    let torpedoCoords = Battlefield.getTorpedoes grid  //Get a list of torpedo coords

    let torpedoCoords =
        List.map (fun (x,y) ->
        let torpedoPerimeter = getNewTorpedoPositions (x,y) grid //Get the perimeter coords of the torpedo
        calculateClosest spyCoords torpedoPerimeter    //Get to coord closest to spy coords from the torpedo perimeter
        ) torpedoCoords

    //Remove old torpedoes
    let updateSector sector x y =
        match sector with
        //If clear, stay clear
        | Clear -> Clear
        //If active, copy sector
        | Active (name, pos, hit) -> Active (name, pos, hit)
        //Remove torpedos
        | Torpedo -> Clear
    let grid = Battlefield.mapGrid updateSector grid

    //Add new torpedoes
    let updateSector sector x y =
        match sector with
        //If clear and position is a new torpedo position, place torpedo
        | Clear when List.contains (x,y) torpedoCoords -> Torpedo
        | Clear -> Clear
        //If active and position is a new torpedo position, hit
        | Active (name, pos, hit) when List.contains (x,y) torpedoCoords -> Active (name, pos, true)
        | Active (name, pos, hit) -> Active (name, pos, hit)
        //Remove torpedos
        | Torpedo -> Clear
    Battlefield.mapGrid updateSector grid
```

interceptNextHit : La stratégie utilisé dans cette fonction est plutôt simple. Je vérifie la vie restante du bateau Cruiser. Si la vie restante est supérieure à 1, j'intercepte les missiles que le Spy lance. Sinon le Cruiser ne l'intercepte pas. De cette façon, le bateau Cruiser reste en vie et il a une chance que le PatrolBoat trouve l'occasion de le réparer.

```
let interceptNextHit (coord: Coord) (grid: Sector Grid) : Sector Grid =
    let cruiserHealth = getRemainingHealth Cruiser grid
    if cruiserHealth > 1 then
        shootAtShip Cruiser grid
    else
        //Hit aimed ship if health is too low
        let updateSector sector x y =
            match sector with
            //If clear
            | Clear -> Clear
            //If active and position, hit
            | Active (name, pos, hit) when (x,y) = coord -> Active (name, pos, true)
            | Active (name, pos, hit) -> Active (name, pos, hit)
            //Remove torpedos
            | Torpedo -> Clear
        Battlefield.mapGrid updateSector grid
```

getPlanePath : Pour cette fonction, j'ai décidé de simplement faire un rectangle du même ratio que le grid. Il est fortement possible de trouver des solutions plus optimales comme la figure 8 mais j'avais de la misère à penser comment le faire. De plus, avec ma gestion de temps médiocre, il m'était plus optimal de me concentrer à travailler sur d'autres fonctions.

```
let createSquarePath grid shrink =
    let (dimx, dimy) = Navigation.getDimsGrid grid

    let minX = shrink
    let maxX = dimx - 1 - shrink
    let minY = shrink
    let maxY = dimy - 1 - shrink

    let top = List.init (maxY - minY + 1) (fun i -> (minX, minY + i))
    let right = List.init (maxX - minX) (fun i -> (minX + 1 + i, maxY))
    let bottom = List.init (maxY - minY) (fun i -> (maxX, maxY - 1 - i))
    let left = List.init (maxX - minX - 1) (fun i -> (maxX - 1 - i, minY))

    top @ right @ bottom @ left

let getPlanePath (grid: Sector Grid) : Coord list =
    createSquarePath grid 1
```

advancePlane : J'ai de la misère imaginer une stratégie pour cette fonction. L'avion fait simplement avance dans son chemin.

```
let advancePlane (stream: Coord Stream) : Coord Stream =
    match stream with
    | Stream.Empty -> Stream.Empty
    | Stream.Cons (_, r) -> r.Force()
```

shiftPath : Pour cette fonction, ma stratégie était de déplacer le chemin vers le Spy dépendamment de sa position sur le grid. Je calcule la position du Spy relatif au centre du grid et je déplace le chemin en conséquent. Par exemple, si le bateau se trouve vers le haut du grid, je déplace le chemin vers le bas, en espérant que le chemin haut du chemin total

croise le Spy. Or, si le Spy est trop aux extrémités, le chemin s'éloigne en conséquent. J'ai eu beaucoup de misère avec cette fonction à cause de la manipulation des flots. C'est ma première expérience avec les flots donc le concept était plus compliqué à comprendre que le reste. Cette fonction est loin d'être ma fonction la plus propre.

```
let rec map f stream =
    match stream with
    | Stream.Empty -> Stream.Empty
    | Cons (head, tail) ->
        let newHead = lazy (f (head.Force()))
        let newRest = lazy (map f (tail.Force()))
        Cons (newHead, newRest)

let shifted = ref (false,North);

let shiftPath (grid: Sector Grid) (stream: Coord Stream) : Coord Stream =
    let spyCoords = getShipCoords Spy grid
    let (dimx, dimy) = Navigation.getDimsGrid grid
    let centerx,centery = (dimx/2, dimy/2)

    //Function to find which direction to shift based on the distance of the spy compared to the center
    let rec shiftDirection spyCoords' biggestDist direction =
        match spyCoords' with
        | [] -> direction
        | (x,y)::rest ->
            let xDist = x - centerx
            let yDist = y - centery
            let biggestDist' = max (abs xDist) (abs yDist)
            let direction' =
                if biggestDist' > biggestDist then
                    if abs(xDist) > abs(yDist) then
                        if xDist > 0 then
                            North
                        else
                            South
                    else
                        if yDist > 0 then
                            West
                        else
                            East
                else
                    direction
            let biggestDist' =
                if biggestDist' > biggestDist then
                    biggestDist'
                else
                    biggestDist
            shiftDirection rest biggestDist' direction'
    let newShiftDirection = shiftDirection spyCoords 0 North

    let isShifted,shiftedDirection = !shifted

    //Shift
    let shift (x,y) =
        shifted := (true, newShiftDirection)
        match newShiftDirection with
        | North -> ((x - 1), y)
        | South -> ((x + 1), y)
        | West -> (x, (y - 1))
        | East -> (x, (y + 1))
```

```
//Unshift
let unshift (x,y) =
    shifted := (false, newShiftDirection)
    match shiftedDirection with
    | North -> ((x + 1), y)
    | South -> ((x - 1), y)
    | West -> (x, (y + 1))
    | East -> (x, (y - 1))

if isShifted then
    map unshift stream
else
    map shift stream
```

revertPath : Ma stratégie pour cette fonction est de comparer la position de l'avion et son prochain avec la position du Spy. Si le prochain est plus proche, on ne renverse pas. Sinon, on renverse le chemin. J'ai eu beaucoup de misère pour les mêmes raisons que la fonction **shiftPath**. De plus, puisque l'avion ne fait pas partir du grid, il était compliqué de trouver comment comparer les positions.

```
let reverted = ref false

let revertPath (grid: Sector Grid) (stream: Coord Stream) : Coord Stream =

    let revert =
        let planePath = getPlanePath grid
        let revertedPath =
            if !reverted then
                planePath
            else
                List.rev planePath
        reverted := not !reverted
        let newStream = cycleList revertedPath
        let isShifted, direction = !shifted
        //Shift
        let shiftMatch (x,y) =
            match direction with
            | North -> ((x - 1), y)
            | South -> ((x + 1), y)
            | West -> (x, (y - 1))
            | East -> (x, (y + 1))

        let shiftMatchedStream =
            if isShifted then
                Stream.map shiftMatch newStream
            else
                newStream

        let rec matchHead s1 s2 =
            match s1, s2 with
            | Cons (head, _), Cons(head2, rest) when head.Force() = head2.Force() -> Cons(lazy head2.Force(), rest)
            | Cons (_,_), Cons(_, rest) -> matchHead s1 (rest.Force())
            | _ -> Stream.Empty //Wont happen

        matchHead stream shiftMatchedStream

    let currentPlanePos = Stream.nth stream 0 //Current plane pos
    let nextPlanePos = Stream.nth stream 1 //Next plane pos

    //Get head of spy
    let rec spyPos coords =
        match coords with
        | (_, pos, _, x, y)::_ when pos = 1 -> (x,y)
        | _::rest -> spyPos rest
        | _ -> (0,0) //Wont happen

    let spyCoords = getShipSectors Spy grid
    let spyHead = spyPos spyCoords

    let distCurrent = calculateDist spyHead currentPlanePos
    let distNext = calculateDist spyHead nextPlanePos

    if distNext >= distCurrent then
        revert
    else
        stream
```

Observations du l'abstraction

La plupart du temps, je réalisais rapidement ce que je devais abstraire pendant que j'écrivais le code. Lorsque je travaille sur une fonction, j'essaye de séparer la fonction en question en plusieurs petites tâches. En faisant cela, je voyais rapidement des ressemblances avec les autres fonctions. Durant le TP1, j'avais créé une fonction du type map pour le grid, ainsi qu'un setSector et getAllSector. Ces fonctions m'ont grandement aidé pour le développement du TP2. De plus, j'ai créé des fonctions encore plus précises qui m'ont beaucoup servi. Par exemple, la fonction getSector m'a aidé à créer la fonction getShipSector, qui à son tour, m'a aidé à créer la fonction getShipCoords. Ce sont des fonctions plutôt simples à mon avis, mais elles sont très utiles.

```
let getShipSectors (name: Name) (grid: Sector Grid) =
    let allSectors = Grid.getAllSector grid getAllActiveSectors  //Get a list of all active sectors
    List.filter (fun (name', _, _, _, _) -> name' = name) allSectors //Get a list of ship sectors

let getShipCoords (name: Name) (grid: Sector Grid) =
    let shipSectors = getShipSectors name grid
    List.map (fun (_, _, _, x, y) -> (x, y)) shipSectors
```

```
//Function to get a list of all sectors in a grid and apply a function to each sector
let getAllSector grid f =
    //Recursive function to iterate through the grid. Store all coords in a list
    let rec checkGrid grid rowIndex =
        match grid with
        //If empty, return empty list
        | Empty -> []
        //If row, check individual sector with getAllActiveSectorRow (returns a list of coords) and concatenate with recursive call on checkGrid
        | Row (sectorList, restGrid) -> (f sectorList rowIndex)@(checkGrid restGrid (rowIndex + 1))
    //Call recursive function
    checkGrid grid 0
```

Note : Dans les étapes pour la vidéo "Perdre un point de vie sur le bateau espion en étant positionnée sous l'avion.", je ne vois pas où il faut coder cette fonction dans le document PDF.