

CS1 (5): テスティング中級編

脇田建

2018.10.22

今日の事例

- ✿ 一昨年、出題した Mandelbrot の顕微鏡の課題のうち、履歴機能（戻るボタン、前へボタン）についてのテストに関して議論します。



Mandelbrot のテストについて

とても興味深い
“自分でやってわかったんですが、~~かなり難しい~~課題でした”

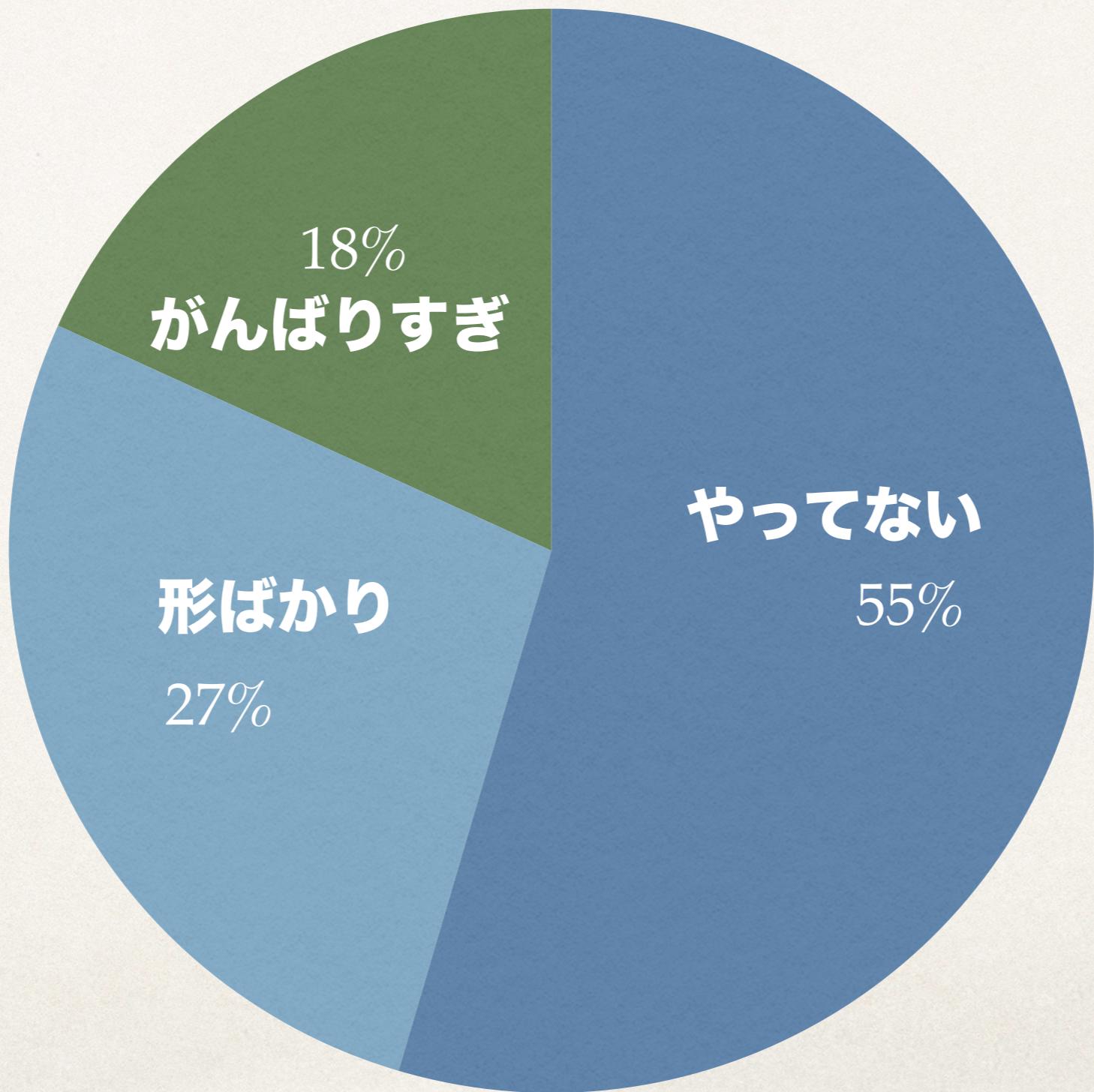
-Ken Wakita

重要な事実？

- ❖ テスト対象からユーザインターフェイス部分を分離しないと scalatest でテストできない
- ❖ object Mandelbrot → object Model + object View
 - ❖ View: ユーザインターフェイスの見栄え
 - ❖ Model: ソフトウェアの心臓部。データ構造、及びデータ構造を操作するメソッドから構成される。
- ❖ (詳しい説明は配布資料を用いて)

過去にMandelbrotの課題に挑んだ学生はどのようにテストしたか？

最初に提出してくれた11名



形ばかり 1

```
+class MandelbrotTest extends FunSuite with Matchers {
+  test("Mandelbrot test") {
+    var memoryTest1: List[(Complex, Complex)] = List((new Complex(-1, -1), new
+    var memoryTest2: List[(Complex, Complex)] = List()
+
+    update(memoryTest1) should be ((new Complex(-1, -1), new Complex(1, 1)))
+    update(memoryTest2) should be ((new Complex(-2, -2), new Complex(2, 2)))
+  }
+
+} ⏪
```

もう少し丁寧なテストを

```
+class ManderbrotTest extends FunSuite with Matchers {
+
+  test("mandelbrot tests"){
+
+    var bl:List[Array[Complex]] = List()
+    var fl:List[Array[Complex]] = List()
+    var region = Array(new Complex(-2, -2), new Complex(2, 2))
+    for(i <- 0 until 10){
+      bl = Array(new Complex(0.1+i/10,0.1+i/10),new Complex(0.3,0.5))::bl
+      fl = Array(new Complex(0.3+i/10,0.2+i/10),new Complex(0.2,0.5))::fl
+    }
+    testBack(region,bl,fl) should equal ((bl.head,bl.tail,bl.head::fl))
+    testForward(region,bl,fl) should equal ((fl.head,fl.head::bl,fl.tail))
+  }
+
+}
```

```
test("backward tests"){
    backwardFutChange(List((new Complex(-1,-1),new Complex(1,1)),(new Complex(-2,-2),new Complex(0,0))),Array((new Complex(-1,-1),new Complex(1,1)),(new Complex(-2,-2),new Complex(0,0)),(new Complex(-1,-1),new Complex(1,1)),(new Complex(-2,-2),new Complex(0,0)))) should equal ((new Complex(-1,-1),new Complex(1,1)),(new Complex(-2,-2),new Complex(0,0)),(new Complex(-1,-1),new Complex(1,1)),(new Complex(-2,-2),new Complex(0,0)))
}
テストの鬼1
}

test("forward tests"){
    forwardFutChange(List((new Complex(-1,-1),new Complex(1,1)),(new Complex(0,0),new Complex(1,1))),Array((new Complex(-1,-1),new Complex(1,1)),(new Complex(0,0),new Complex(1,1)),(new Complex(0,0),new Complex(1,1)),(new Complex(-1,-1),new Complex(1,1)))) should equal ((new Complex(-1,-1),new Complex(1,1)),(new Complex(0,0),new Complex(1,1)),(new Complex(0,0),new Complex(1,1)),(new Complex(-1,-1),new Complex(1,1)))
}
}

test("favorite test"){
    favAdd(List((new Complex(-1,-1),new Complex(1,1))),Array(new Complex(0,0),new Complex(1,1))) should equal ((new Complex(-1,-1),new Complex(1,1)),(new Complex(0,0),new Complex(1,1)))
}
}

test("up test"){
    regionUp(Array(new Complex(-1,-1),new Complex(1,1))) should equal ((new Complex(-1,-1.2),new Complex(0,0)),(new Complex(-1,-1),new Complex(1,1)))
}
}

test("down test"){
    regionDown(Array(new Complex(-1,-1),new Complex(1,1))) should equal ((new Complex(-1,-0.8),new Complex(0,0)),(new Complex(-1,-1),new Complex(1,1)))
}
}

test("right test"){
    regionRight(Array(new Complex(-1,-1),new Complex(1,1))) should equal ((new Complex(-0.8,-1),new Complex(0,0)),(new Complex(-1,-1),new Complex(1,1)))
}
}

test("left test"){
    regionLeft(Array(new Complex(-1,-1),new Complex(1,1))) should equal ((new Complex(-1.2,-1),new Complex(0,0)),(new Complex(-1,-1),new Complex(1,1)))
}
}
```

```
def LACequal(lst1>List[Array[Complex]], lst2>List[Array[Complex]]): Boolean = {
  (lst1, lst2) match {
    case (a1::rest1, a2::rest2) => if(a1(0) == a2(0) && a1(1) == a2(1)) LACequal(rest1, rest2) else
      case (Nil, Nil) => true
      case (_, _) => false
  }
}

calcforward(region, bList, fList) match {
  case (newregion, newbList, newfList) =>
    LACequal(List(newregion), List(Array(new Complex(-0.3,0.2), new Complex(0.7,0.5)))) should equal
    LACequal(newbList, List(Array(new Complex(-0.3,0.2), new Complex(0.7,0.5))), Array(new Complex(-1
    LACequal(newfList, Nil) should equal (true)

  region = newregion
  bList = newbList
  fList = newfList
}

calcbck(region, bList, fList) match {
  case (newregion, newbList, newfList) =>
    LACequal(List(newregion), List(Array(new Complex(-1,-1), new Complex(1,1)))) should equal (true)
    LACequal(newbList, List(Array(new Complex(-1,-1), new Complex(1,1))), Array(new Complex(-2,-2), n
    LACequal(newfList, List(Array(new Complex(-0.3,0.2), new Complex(0.7,0.5)))) should equal (true)

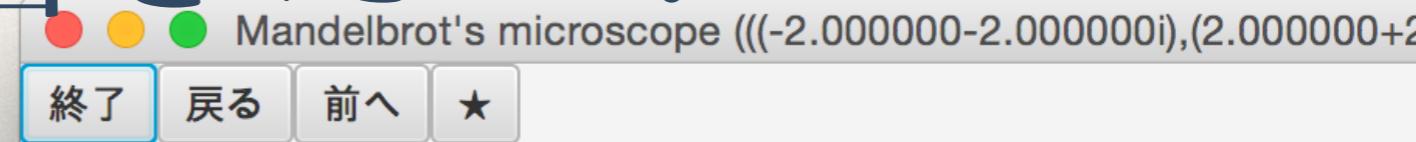
  region = newregion
  bList = newbList
  fList = newfList
}
```

テストの鬼 2

問題点：気持ちよくテストしたい

- ＊ テストをすることでソフトウェアの完成度についての信頼感を得たい。
- ＊ 不完全なテストでは、信頼性を担保できない。
- ＊ テストに用いる値についての自信がないとテストの正しさについての不安に苛まれる。
- ＊ テストの実施を容易にしたい
- ＊ テストを簡潔に記述したい。
- ＊ テストに用いる値について、（テスター本人だけでなく）誰が見ても、速やかに正しさが納得できるようなものを用いたい。

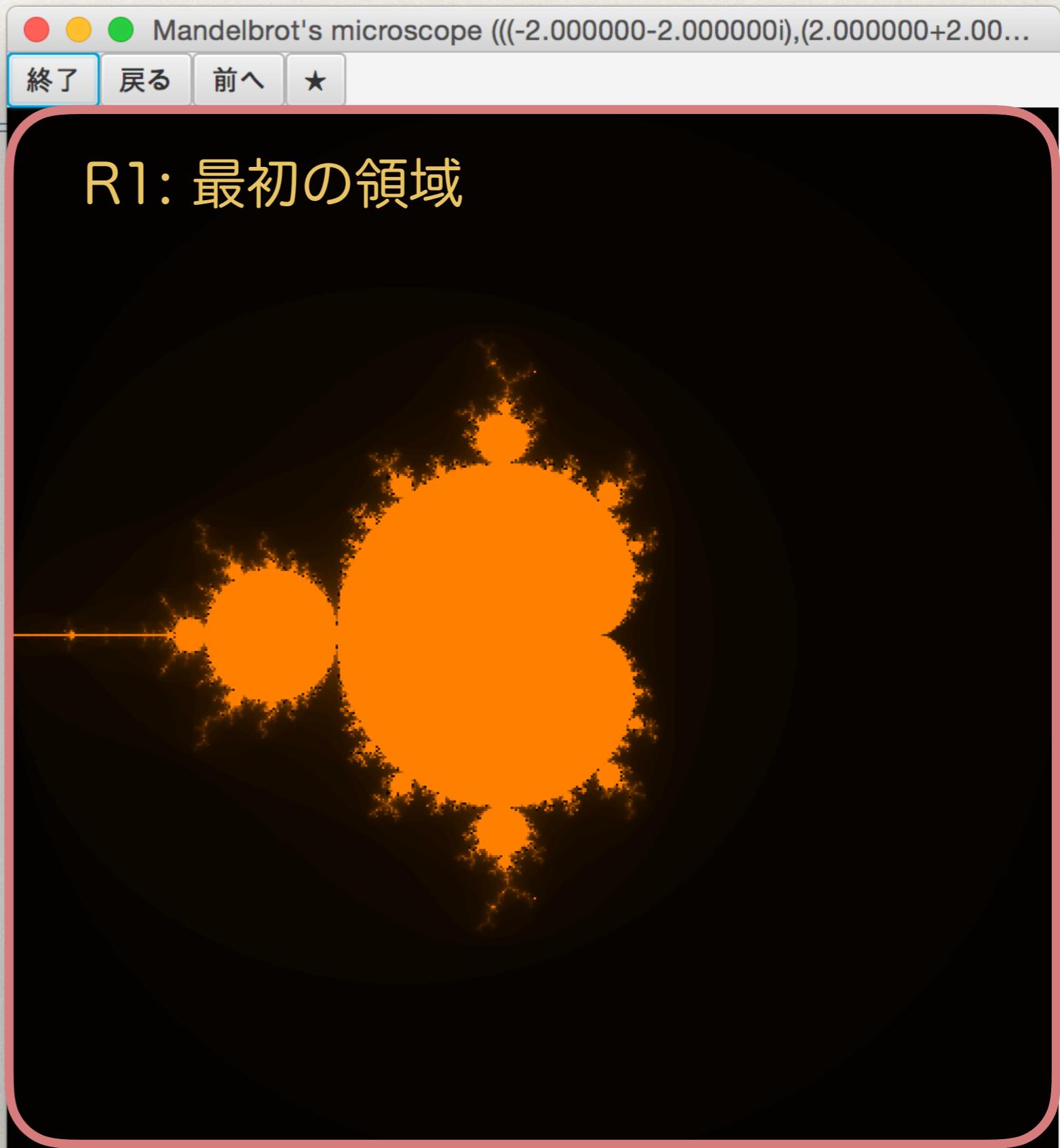
どんなテストを書きたい？



- ❖ どれだけのテスト項目を記述すれば、自分のソフトの完成度についての信頼が得られるだろう。

データの変化を 見てみよう

- ❖ 最初: (R1)



データの変化を 見てみよう

- ❖ 最初: (R1)
- ❖ 拡大 → (R2, R1)



データの変化を 見てみよう

- ❖ 最初: (R1)
- ❖ 拡大 → (R2, R1)
- ❖ 拡大 → (R3, R2, R1)



一步戻る

- ❖ 最初: (R1)
- ❖ 拡大 → (R2, R1)
- ❖ 拡大 → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
将来の進むに備えて,
R3を覚えつつ, 表示する
領域も記憶する



もう一步戻る

- ❖ 最初: (R1)
- ❖ 拡大 → (R2, R1)
- ❖ 拡大 → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)



もう戻れない

- ❖ 最初: (R1)
- ❖ 拡大 → (R2, R1)
- ❖ 拡大 → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)



進む

- ✿ 最初: (R1)
- ✿ 拡大 → (R2, R1)
- ✿ 拡大 → (R3, R2, R1)
- ✿ 戻る → (R3, R2, R1)
- ✿ 戻る → (R3, R2, R1)
- ✿ 戻る → (R3, R2, R1)
- ✿ 進む → (R3, R2, R1)



もう一步進む

- ❖ 最初: (R1)
- ❖ 拡大 → (R2, R1)
- ❖ 拡大 → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 進む → (R3, R2, R1)
- ❖ 進む → (R3, R2, R1)



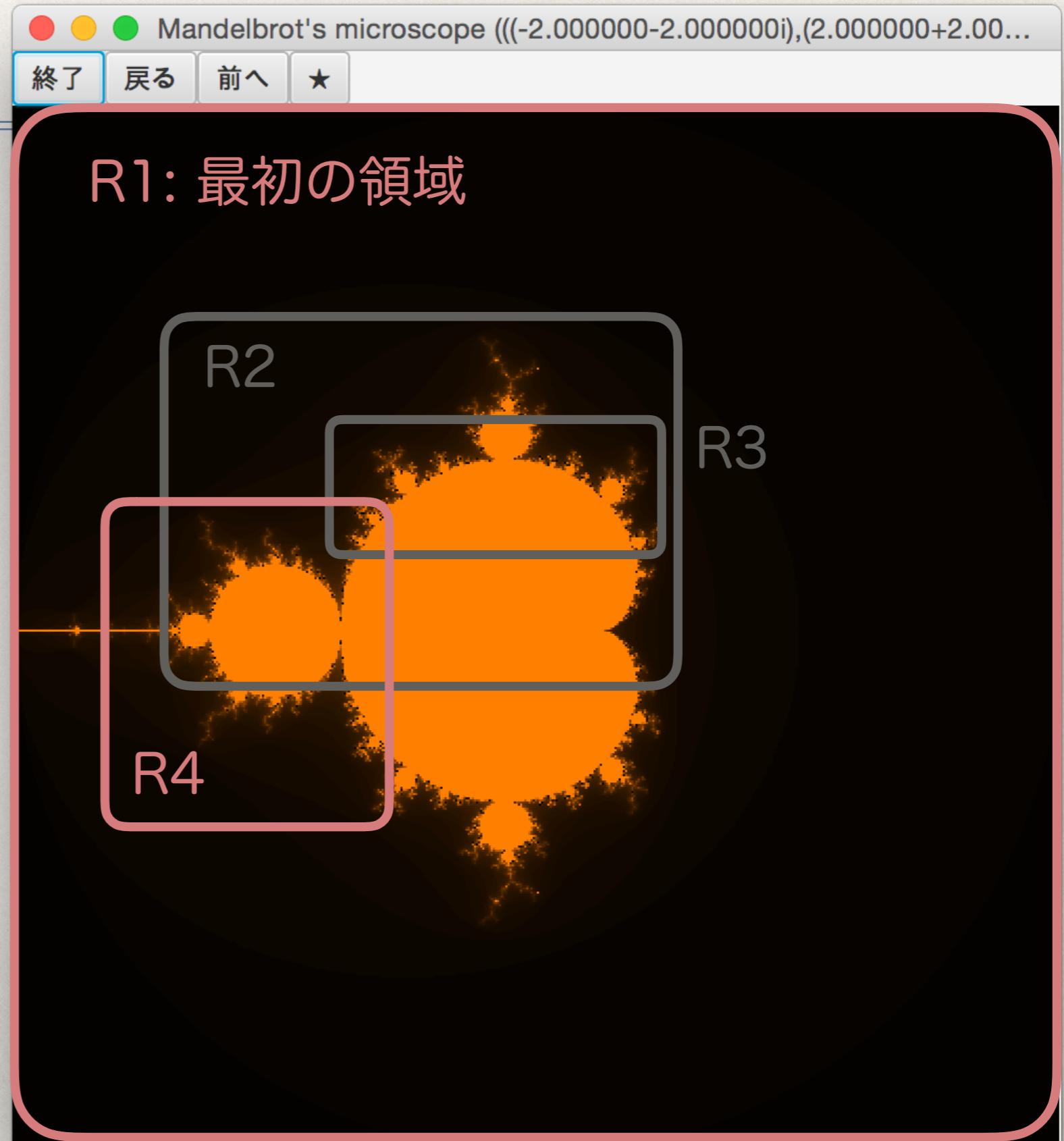
もう進めない

- ❖ 最初: (R1)
- ❖ 拡大 → (R2, R1)
- ❖ 拡大 → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 戻る → (R3, R2, R1)
- ❖ 進む → (R3, R2, R1)
- ❖ 進む → (R3, R2, R1)
- ❖ 進む → (R3, R2, R1)



ここで拡大

- ✿ ... → (R₃, R₂, R₁)
- ✿ 拡大 → (R₄, R₁)



履歴管理に関するテスト

現象の分析	テストの内容
最初: (R1)	S0 should be (R1)
拡大 → (R2 , R1)	S1 = Drag (S0) should be (R2 , R1)
拡大 → (R3 , R2, R1)	S2 = Drag (S1) should be (R3 , R2, R1)
戻る → (R3, R2 , R1)	S3 = 戻る (S2) should be (R3, R2 , R1)
戻る → (R3, R2, R1)	S4 = 戻る (S3) should be (R3, R2, R1)
戻る → (R3, R2, R1)	S5 = 戻る (S4) should be (R3, R2, R1)
進む → (R3, R2 , R1)	S6 = 進む (S5) should be (R3, R2 , R1)
進む → (R3 , R2, R1)	S7 = 進む (S6) should be (R3 , R2, R1)
進む → (R3 , R2, R1)	S8 = 進む (S7) should be (R3 , R2, R1)

では、どうやってテストケースを書くの？

- ❖ lx05 リポジトリ (=配布資料) 参照

test/mandelbrot1.scala

- T("DDBDB", B(hDDBD), 3, 1) $S3 = \text{Bwd}(S2)$ should be (R3, **R2**, R1)
- 拡大, 拡大, Bwd, 拡大, Bwdしたら, 履歴の大きさは3で, 表示されるのは履歴の1番目の要素
- D: 拡大、B: Bwd (Backward; 戻る) のつもり
- T(テストのラベル,
次の状態 = 操作(前の状態)、
次の状態における履歴の大きさ、
履歴中のどの箇所を表示するか)

test/mandelbrot1.scala

- ✿ もうひとつの工夫：テスト用の補助関数(T)を定義

```
def T(label: String, h: History, size: Int, pos: Int) {  
    test(f"$label: 履歴の長さが仕様と合致すること") {  
        h._1.length should be (size)  
    }  
  
    test(f"$label: 現在、表示している箇所が履歴情報の該当位置  
        h._2 should be (pos)  
    }  
}  
  
val h0: History = history  
T("h0", h0, 1, 0)  
T("B(h0)", B(h0), 1, 0)  
T("F(h0)", F(h0), 1, 0)
```

補助関数のおかげで
テストの記述が簡潔になる

test/mandelbrot1.scala

- T("DDBDB", B(hDDBD), 3, 1) $S_3 = \text{Bwd}(S_2)$ should be (R3, **R2**, R1)
- 拡大, 拡大, Bwd, 拡大, Bwdしたら, 履歴の大きさは3で, 表示されるのは履歴の1番目の要素
- D: 拡大、B: Bwd (Backward; 戻る) のつもり
- T(テストのラベル,
次の状態 = 操作(前の状態)、
次の状態における履歴の大きさ、
履歴中のどの箇所を表示するか)

このテスト記述の難点
3, 1 の意味が把握しにくい。
正しさをパッと見て判断し
にくい。

test/mandelbrot2.scala

S3 = **Bwd**(S2) should be (R3, **R2**, R1)

- ❖ T("DDBDB", B(hDDBD), "_X_")
- ❖ 拡大, 拡大, Bwd, 拡大, Bwdしたら, 履歴の大きさは3で, 表示されるのは履歴の第二要素
- ❖ 履歴データの状況をわかりやすく表現することで、テストの可読性を向上した。

test/mandelbrot2.scala

```
val hDDB = B(hDD)
T("hDDB",          hDDB,           "X_")
T("F(hDDB)",       F(hDDB),        "X_")
T("F(F(hDDB))",   F(F(hDDB)),    "X_")
T("B(hDDB)",       B(hDDB),        "X")
T("B(B(hDDB))",   B(B(hDDB)),    "X")
```

```
val hDDBD = D(hDDB)
T("hDDBD",          hDDBD,          "X_")
T("F(hDDBD)",       F(hDDBD),       "X_")
T("B(hDDBD)",       B(hDDBD),       "X_")
T("B(B(hDDBD))",   B(B(hDDBD)),   "X_")
T("B(B(B(hDDBD))))", B(B(B(hDDBD))), "X")
```

このテスト記述の難点
第一引数と第二引数が冗長

test/mandelbrot3.scala

```
val specDDB: Spec = B(specDD)
T(F(specDDB),      "X__")
T(F(F(specDDB))), "X__"

val specDDBD: Spec = D(specDDB)
T(specDDBD,          "X__")
T(F(specDDBD),      "X__")
T(B(specDDBD),      "_X_")
T(B(B(specDDBD))), " _X"
T(B(B(B(specDDBD)))), "__X"
```

- ❖ $T(B(specDDBD), “_X_”)$
- ❖ ラベルと履歴の構造を同時に構成するような補助関数 D, B, F などを定義。

test/mandelbrot3.scala

```
type Spec = (String, History)

def D(spec: Spec) = {
    val h = spec._2
    (f"D(${spec._1})", subRegion(h, rClick(h), rClick(h)))
}

def B(spec: Spec) = {
    val h = spec._2
    (f"B(${spec._1})", backward(h))
}

def F(spec: Spec) = {
    val h = spec._2
    (f"F(${spec._1})", forward(h))
}
```

test/mandelbrot3.scala

```
val specDDB: Spec = B(specDD)
T(F(specDDB),      "X_")
T(F(F(specDDB))), "X_")
```

```
val specDDBD: Spec = D(specDDB)
T(specDDBD,          "X_")
T(F(specDDBD),       "X_")
T(B(specDDBD),       "X_")
T(B(B(specDDBD))),  "X_")
T(B(B(B(specDDBD)))), "X_")
```



このテスト記述の難点
もっと簡潔にテストを記述で
きないか？

test/mandelbrot4.scala

```
T("DDBFF", "X_,X__,_X_,X__,X__")
T("DDBBB", "X_,X__,_X_,_X,X,__X")
```

```
T("DDBDF", "X_,X__,_X_,X__,X__")
T("DDBDBBBB", "X_,X__,_X_,X__,_X_,_X,X,__X")
```

- T("DDBDBBBB", "X_,X__,_X_,X__,_X_,_X,X,__X")
- D=拡大, D=拡大, B=戻る, D=拡大, B=戻る, ... t
という遷移に沿って一気にたくさんのテストを実施すればいいじゃないか。

test/mandelbrot4.scala

```
def T(specs: String, answers: String) {  
    var spec1 = (f".", history) // 初期状態：空の履歴を表す  
    for ((command, answer) <- specs.split("\n").zip(answers.split("\n"))) {  
        // spec2は、現時点(spec1)において command を実行したときの状態とする  
        val spec2 = update(command, spec1)  
        T(spec1, spec2, answer) // テストの実施  
        spec1 = spec2 // spec2に状態を遷移させる  
    }  
}
```

```
T("DDBDF", "X_,X__,_X_,X__,X__")  
T("DDBDBBB", "X_,X__,_X_,X__,_X_,_X,__X")
```

森先生とTAの講評

- ✿ 履歴情報の領域の変化は確かにテストされています
- ✿ でも、履歴情報に含まれている座標についてはテストしていないんじゃないですか？

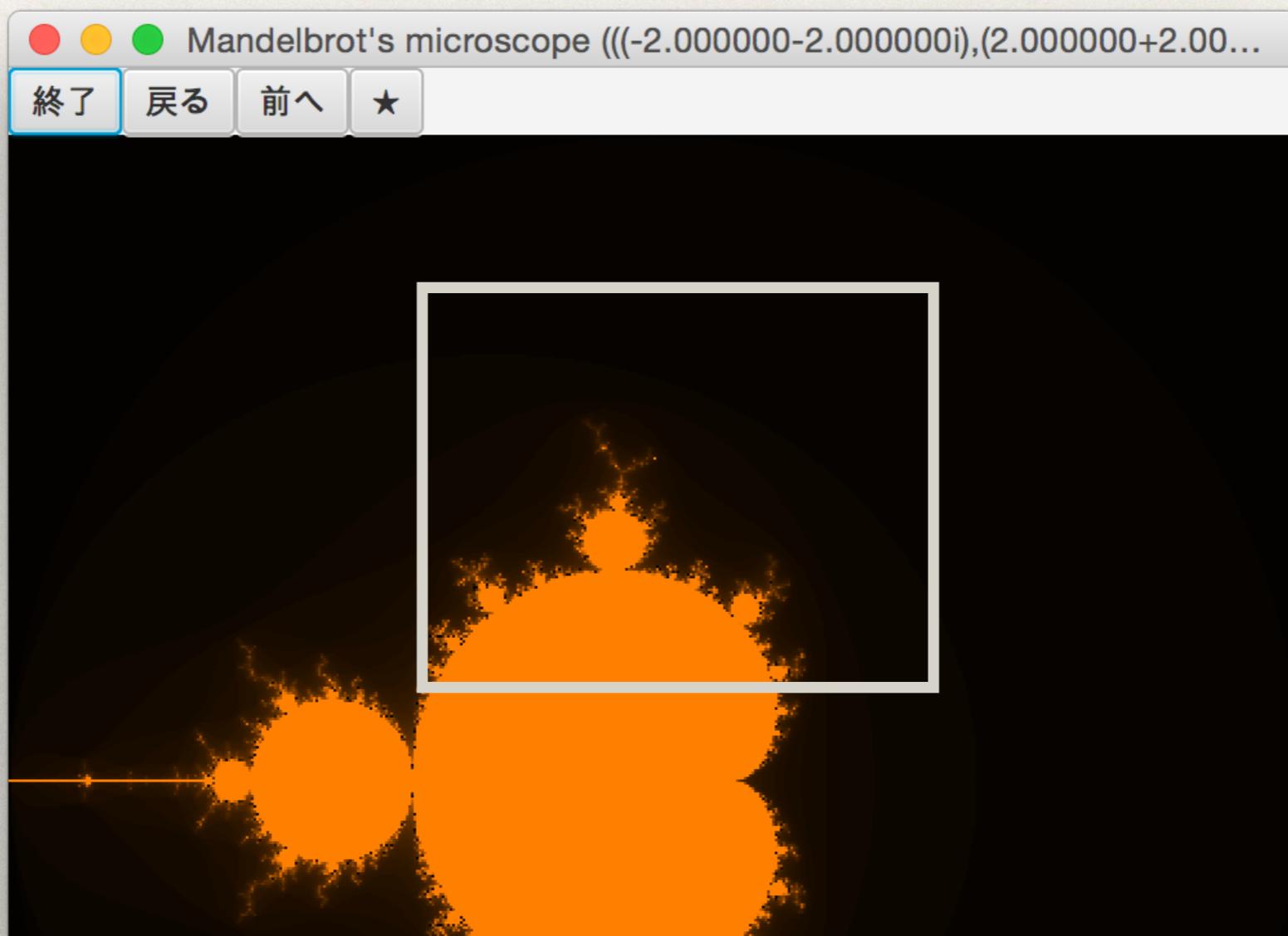
たとえば

- ❖ ドラッグで表示画面の内側を選択しているのに、実際にはその外側が表示されてはいないだろうか？
- ❖ (実際に最初に書いたコードにはこのバグがあった。
後述)
- ❖ ドラッグで表示された画面情報以外の履歴情報は本当に破壊されていないだろうか？

表示領域の包含関係についての テスト

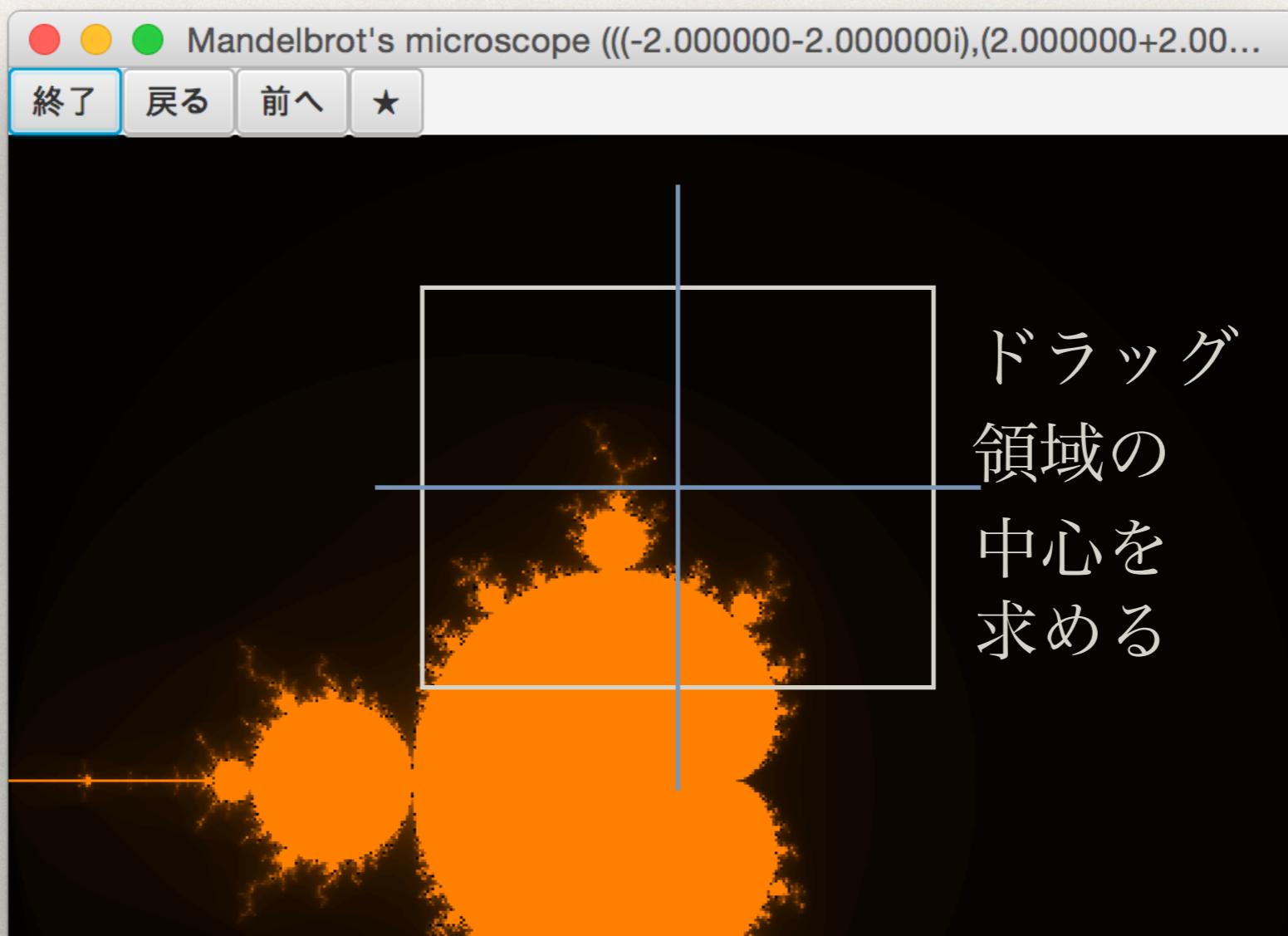
脇田のバグつきアルゴリズム

- ドラッグで表示画面の内側を選択しているのに、実際にはその外側が表示されてはいないだろうか？



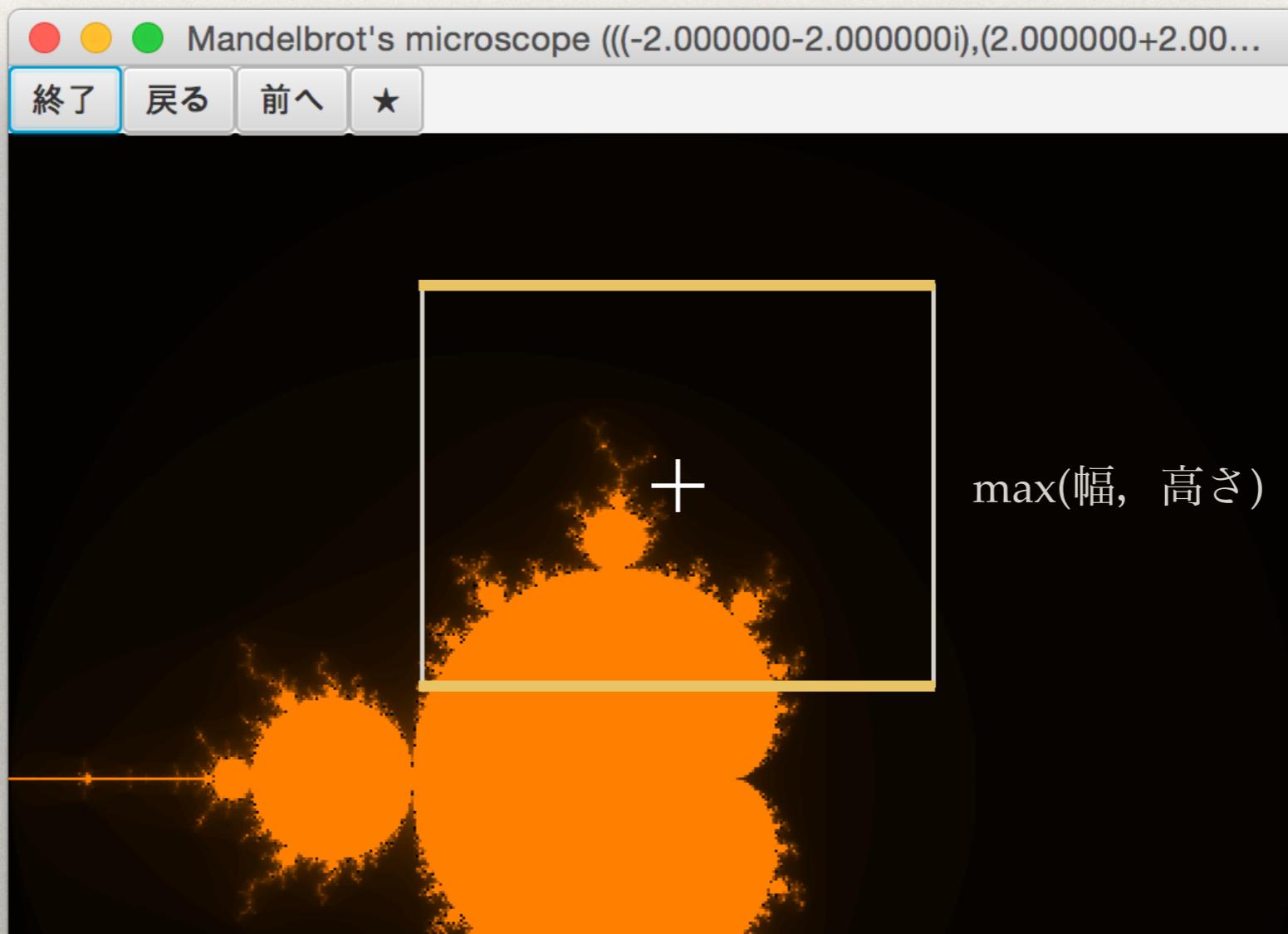
脇田のバグつきアルゴリズム

- ドラッグで表示画面の内側を選択しているのに、実際にはその外側が表示されてはいないだろうか？



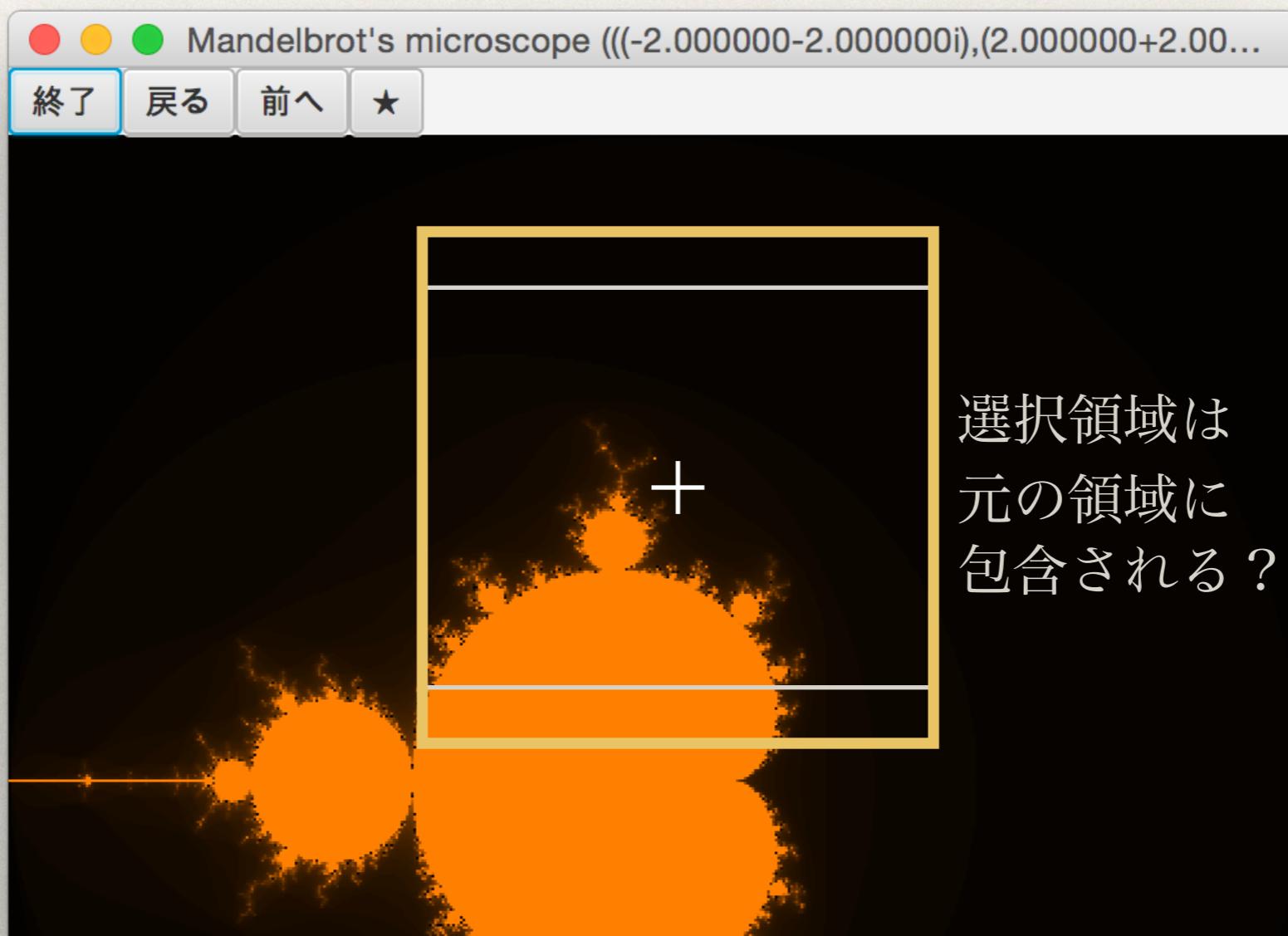
脇田のバグつきアルゴリズム

- ドラッグで表示画面の内側を選択しているのに、実際にはその外側が表示されてはいないだろうか？



脇田のバグつきアルゴリズム

- ドラッグで表示画面の内側を選択しているのに、実際にはその外側が表示されてはいないだろうか？



脇田のバグつきコード

```
46  def subRegion(h: History, p1: Complex, p2: Complex) = {
47    val size = max(abs(p2.re - p1.re), abs(p2.im - p1.im))
48    val c1 = new Complex((p1.re + p2.re - size) / 2, (p1.im + p2.im - size) / 2)
49    val c2 = new Complex((p1.re + p2.re + size) / 2, (p1.im + p2.im + size) / 2)
50    ((c1, c2) :: dropToCurrent(h), 0)
51 }
```

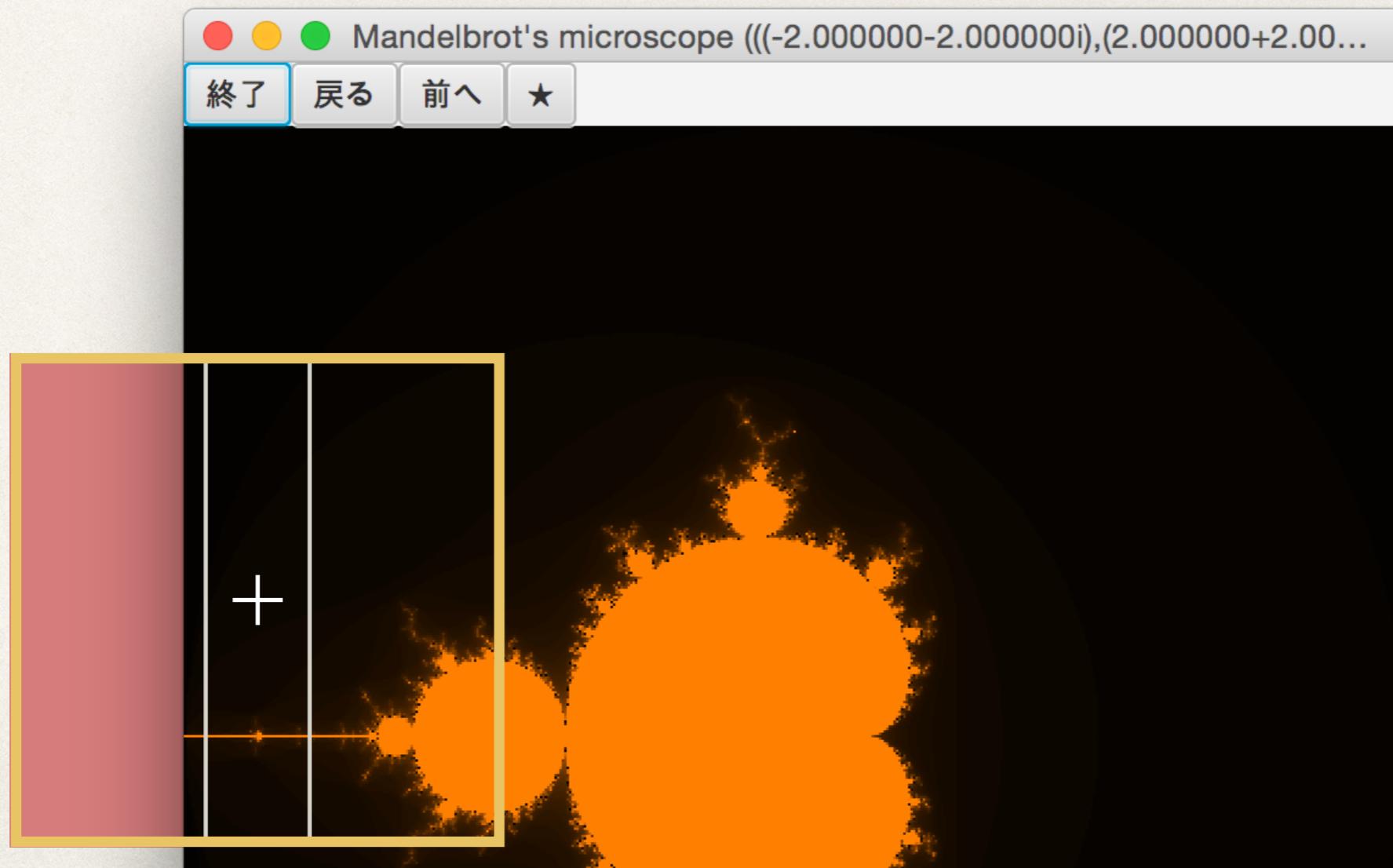
テストケース

```
test(f"$label: 履歴情報に含まれる表示範囲について包含関係が成立
// すなわち、履歴情報: [ Rn, ..., R2, R1 ] のとき  $Rn \subsetneq ...$ 
  for (i <- 0 to h2.length - 2) {
    (h2(i), h2(i+1)) match {
      case ((c21, c22), (c11, c12)) => {
        List(c11.re, c21.re, c22.re, c12.re) shouldBe sorted
        List(c11.im, c21.im, c22.im, c12.im) shouldBe sorted
      }
    }
  }
}
```

- なんと、すべてのテストケースが失敗！

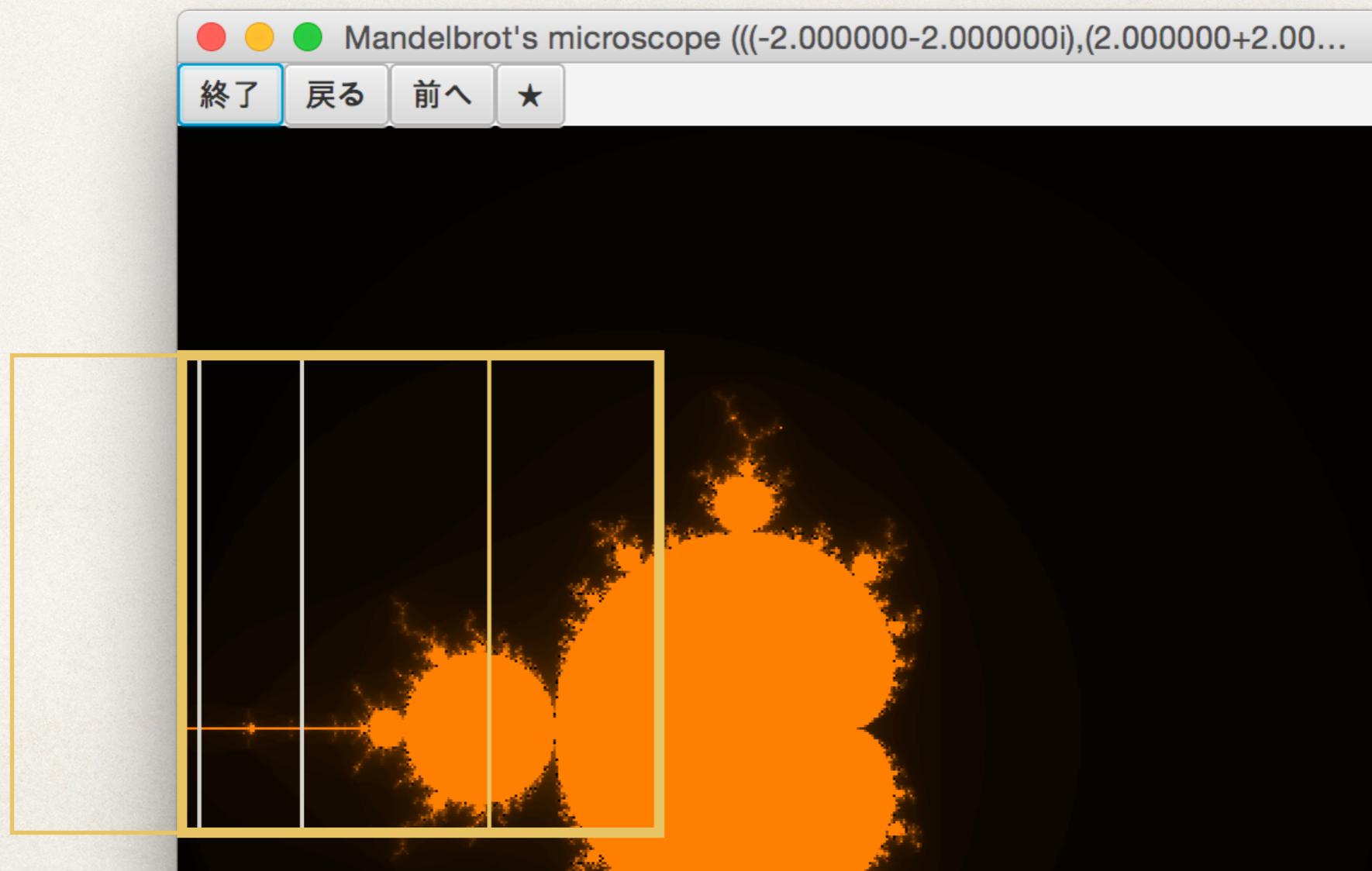
脇田のバグの種あかし

- 表示画面のすぐ内側の細長い領域が選択されると、表示画面からはみだした領域が選択される。



アルゴリズムの修正

- ＊ はみだしそうになつたら、元の表示画面内に押し込む。



テストケースに併せてバグの修正

```
def subRegion(h: History, p1: Complex, p2: Complex) = {
    currentRegion(h) match { case (c1, c2) =>
        val size = max(abs(p2.re - p1.re), abs(p2.im - p1.im))
        val _re1 = (p1.re + p2.re - size) / 2
        val _im1 = (p1.im + p2.im - size) / 2

        val re1 = min(max(c1.re, _re1), c2.re - size)
        val im1 = min(max(c1.im, _im1), c2.im - size)

        ((new Complex(re1, im1), new Complex(re1 + size, im1 + size)) ::

         dropToCurrent(h), 0)
    }
}
```

履歴を行き来するときに履歴情報
は不変か？

新たな検査仕様

- ✿ 履歴情報のうち現在表示されたもの x or X
 - ✿ X (大文字) 直前の操作で履歴に追加された新規データ
 - ✿ x, _: 直前の操作で変化しない
- ✿ D(rag)

```
T("DDBFF", "X_,X__,_X_,X__,X__")
T("DDBBB", "X_,X__,_X_,_X,X,__X")
```
- ✿ F(orward)

```
T("DDBDF", "X_,X__,_X_,X__,X__")
T("DDBDBBB", "X_,X__,_X_,X__,_X,X,__X")
```
- ✿ B(ackward)

履歴情報のうちドラッグされたばかり の領域(X)のみが更新されること

```
test(f"$label: 履歴情報のうちドラッグされたばかりの領域(X)のみが更新されること") {
    val scenes = answer.split("")
    val (len1, len2) = (h1.length, h2.length)
    for (s <- scenes.indices) {
        val i = len2 - 1 - s
        if (i < len1 - 1) {
            val unchanged = (h2(s) == h1(len1 - 1 - i))
            unchanged should be(scenes(s) != 'X')
        }
    }
}
```



```
// 履歴情報のうちドラッグされたばかりの領域(X)のみが変更されていること
val scenes = answer.split("")
val len1 = h1.length
val len2 = h2.length
for (s <- scenes.indices) {
    val i = len2 - 1 - s          オブジェクトの同一性判定
    if (i < len1 - 1) {
        val unchanged = (h2(s) == h1(len1 - 1 - i))
        unchanged should be (scenes(s) != 'X')
    }
}
```



Mandelbrotのテスティング

- ❖ テスト記述量10件、テスト項目5種類
- ❖ 履歴情報の大きさが仕様に合致すること
- ❖ 履歴情報における表示画面位置(履歴情報の第二成分)が仕様の'[Xx]'の出現位置に合致すること
- ❖ 履歴の底は初期画面であること
- ❖ 履歴情報に含まれる表示範囲について包含関係が成立すること。
- ❖ 履歴情報のうちドラッグされたばかりの領域(X)のみが変更されていること

Mandelbrotのテスティング

✿ テスト記述量10件、テスト項目5種類、テスト総数185

```
[info] - 35 (.DDBDB->.DDBDBB): 履歴情報に含まれる表示範囲について包含関係
[info] - 35 (.DDBDB->.DDBDBB): 履歴情報のうちドラッグされたばかりの領域(初期)
[info] - 36 (.DDBDBB->.DDBDBBB): 履歴の長さが仕様と合致すること
[info] - 36 (.DDBDBB->.DDBDBBB): 履歴情報における表示画面位置(履歴情報の上)
[info] - 36 (.DDBDBB->.DDBDBBB): 履歴の底は初期画面であること
[info] - 36 (.DDBDBB->.DDBDBBB): 履歴情報に含まれる表示範囲について包含関係
[info] - 36 (.DDBDBB->.DDBDBBB): 履歴情報のうちドラッグされたばかりの領域(上)
[info] ScalaTest
[info] Run completed in 1 second, 594 milliseconds.
[info] Total number of tests run: 185
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 185, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[info] Passed: Total 185, Failed 0, Errors 0, Passed 185
[success] Total time: 6 s, completed 2016/10/28 12:02:57
```

Mandelbrotテストのまとめ

- ✿ ソフトウェアのテスト作業は想像力と創造性をともに要求される高度に知的なパズル
- ✿ 次回、授業の前に小テスト
- ✿ 今回のテスティングに関してよく復習しておいて下さい

Mandelbrotテストのまとめ

- ✿ ソフトウェアのテスト作業は想像力と創造性とともに要求される高度に知的なパズル
- ✿ Mandelbrotでお腹いっぱいになつたので、近所の八百屋で見つけたロマネスコをいただきました。
- ✿ デザートが欲しい人はここが面白いですよ。