CS1 (7): 続，状態

脇田建

2015.12.1

Examples of Memory Usage (HtDP 37)

事例１ 色あてクイズ

クイズのやり方

- 二人プレーヤ
  - 親 – 相手にわからないように2つの領域の色を定める
  - 挑戦者 – . . .

状態の初期化について

- 色あてクイズ

挑戦者の回答

C1　C2

親がひそかに正解を設定

C1　C2

色のリスト

クイズのやり方

- 二人プレーヤ
  - 親 – 相手にわからないように2つの領域の色を定める
  - 挑戦者 – それぞれの領域の色の組を言い当てる．何度でも試行できるが，試行回数が少ないほどよい．

挑戦者の回答

- 色あてクイズ

挑戦者の回答

C1　C2

親が設定した正解

C1　C2

色のリスト

挑戦者の回答に対する親の応答

1. Perfect! – 大当たり．2つの領域の色をそれぞれ完全に言いあてた
2. OneColorAtCorrectPosition – いずれか一方の領域の色を正しく答えた
3. OneColorOccurs – いずれかの予想が正解の色だが，場所の予想は間違っている
4. NothingCorrect – まるで見当外れ

挑戦者の回答へのマスターの返事

- 色あてクイズ

3rd trial: 1 color occurs

リスト

挑戦者の回答

C1　C2

ゲームマスターが設定した正解

C1　C2

2つの機能

- 親: 2つの領域の色を設定する機能
- 子の予想について正しく応答する機能
  - 正解と回答の照合
  - 試行回数の記録

2つの機能 → 関数

機能	関数
✤ 2つの領域の色を設定する機能	master
✤ プレーヤの予想について正しく応答する機能	master_check

ゲームの進行例

1. master() – 2つの領域への色の指定(赤, 黒)
2. master_check(白, 青) → NothingCorrect
3. master_check(桃, 赤) → OneColorOccurs
4. …
5. master_check(赤, 黒) → Perfect!
6. master() – 新しいゲームの開始．2つの領域への色の組を指定(白, 青)
7. master_check(白, 青) → Perfect!
8. master() – 新しいゲームの開始.

ゲームの進行例

1. **master()** – 2つの領域への色の指定(赤, 黒)
2. master_check(白, 青) → NothingCorrect
3. master_check(桃, 赤) → OneColorOccurs
4. …
5. master_check(赤, 黒) → Perfect!
6. **master()** – 新しいゲームの開始．2つの領域への色の組を指定(白, 青)
7. master_check(白, 青) → Perfect!
8. **master()** – 新しいゲームの開始.

master_checkの特徴

関数ではない

1. master() – 2つの領域への色の指定(赤, 黒)
2. **master_check(白, 青) → NothingCorrect**
3. master_check(桃, 赤) → OneColorOccurs
4. …
5. master_check(赤, 黒) → Perfect!
6. master() – 新しいゲームの開始．2つの領域への色の組を指定(白, 青)
7. **master_check(白, 青) → Perfect!**
8. master() – 新しいゲームの開始.

master/master_check の特徴分析より，設計案 1

- master は master_check に状態を通して影響を及ぼす
- 非関数的な影響 → 状態 ～ 状態変数
  - 状態変数 trial: Int
  - 状態変数 colorL: Color, colorR: Color

    Color = Black, White, Red, Blue, Green, Gold, Pink, Orange, Purple, or Navy

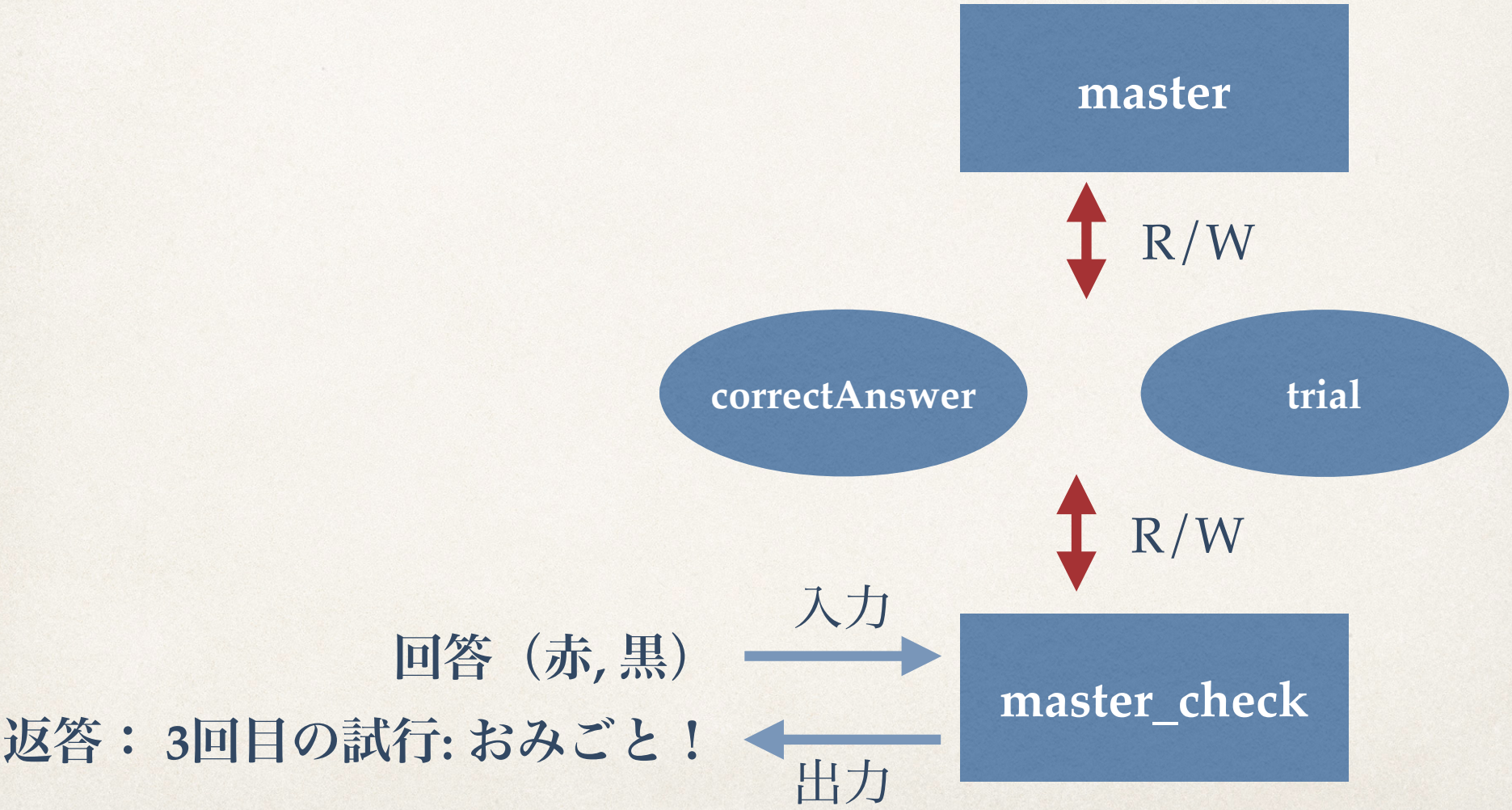
master/master_check の特徴分析より，設計案２

- master は master_check に状態を通して影響を及ぼす
- 非関数的な影響 → 状態 〜 状態変数
  - 状態変数 trial: Int
  - 状態変数 correctAnswer: (Int, Int)

    Color = Int / / 1 => Black, 2 => White, 9 => Navy

状態を用いるデザインレシピの適用

- 問題領域の→ 機能の分析と命名
- 機能の詳細の分析 → 状態の存在が明らかに
- 状態変数の命名 → correctAnswer
  - 状態変数が保存しうる値について分析 → 型
  - 初期化，値の更新方法
  - var correctAnswer: List[Int] = List(0, 0) / / 黒, 黒


master
R/W
correctAnswer
trial
R/W
入力
回答（赤, 黒）
master_check
返答： 3回目の試行: おみごと！
出力

作例: lx07.ColorQuiz

事例２　Hangman

Hangman

- 言葉あてクイズ
  - 親は予め言葉を選んでおく
  - 子はその言葉に使われている文字を予想する
  - 親は子の予想を受けてその正しさを答える

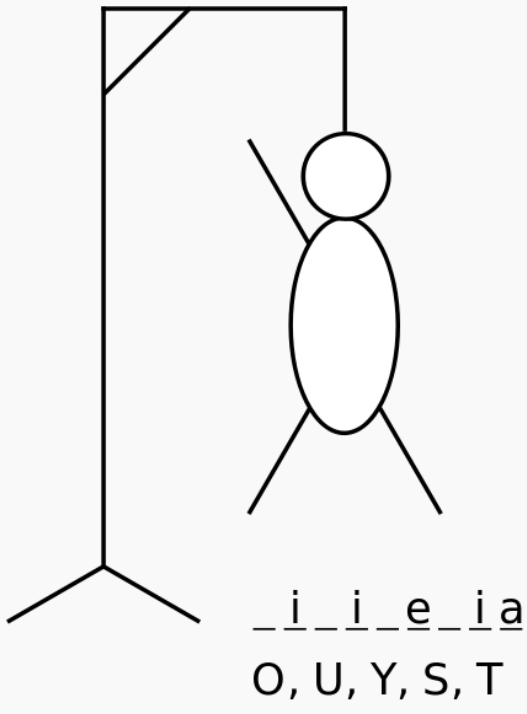
ハングマン（ゲーム）@Wikipedia

遊び方 [編集]

まず。出題者と解答者を決める。

1. 出題者は出題する単語を選び、その単語の文字数を表す下線を引く。絞首台を描く。
2. 解答者は、単語に入っていると思われるアルファベットを一つ答える。
3. 出題者はアルファベットが回答の単語に含まれているか判定する。
   - アルファベットが単語に含まれているならば、下線の上のその文字が入る場所すべてにその文字を書く。
   - アルファベットが単語に含まれていないならば、絞首台につるされる人の絵を描き加える。
4. 勝敗が決まるまで2.3.を繰り返す。以下のときに勝敗は決まる。
   - 解答者が単語を正解する。-解答者の勝利
   - 絞首台の人の絵が完成する。-出題者の勝利

最終的にこの絵は「吊るされた男」になるように描かれる。絵が不適切ではないか何度か議論になっているが、今日でも同じような絵が用いられる。



途中経過の一例

機能

機能	関数
（親が）単語を定める	hangman
子の予想を受けて，予想の正しさを答える	hangmanGuess

ゲームの進行例

- hangman
- hangmanGuess(‘a’)
  → Sorry (head): _ _ _ _ _ _ _
- hangmanGuess(‘i’)
  → Good guess! _ _ _ _ i _
- hangmanGuess(’s’)
  → Good guess! s _ _ _ i _
- hangmanGuess(‘i’)
  → Sorry (body): s _ _ _ i _
- …
- hangmanGuess(‘d’)
  → You wan!
- hangman
- hangmanGuess(‘a’)
  → Good guess! _ _ _ _ _ a m

ゲームの進行例

- hangman – stupid
- hangmanGuess(‘a’)
  → Sorry (head): _ _ _ _ _ _ _
- hangmanGuess(‘i’)
  → Good guess! _ _ _ _ i _
- hangmanGuess(’s’)
  → Good guess! s _ _ _ i _
- hangmanGuess(‘i’)
  → Sorry (body): s _ _ _ i _
- …
- hangmanGuess(‘d’)
  → You wan!
- hangman
- hangmanGuess(‘a’)
  → Good guess! _ _ _ _ _ a _

まずはゲームをやってみよう

- A君はある英単語を選びました.
- B君は単語に使われている文字をあてて下さい.

子の回答に対する応答パターン

- hangman – program
- hangmanGuess('k')
  → Sorry (head): _ r _ _ r _ _
- hangmanGuess('a')
  → Good guess! _ r _ _ r a _
- hangmanGuess('m')
  → You wan!

	描画する部分	予想のあたった文字
Sorry	head	_ r _ _ r _ _
Good Guess!		_ r _ _ r a _
You wan!		
The End		p r o g r a m

Hangmanの履歴依存性

- hangmanGuess ‘a’
  → Sorry (head): _ _ _ _ _ _ _
- hangmanGuess ‘i’
  → Good guess!: _ _ _ _ i _
- hangmanGuess ‘s’
  → Good guess!: s _ _ _ i _
- hangmanGuess ‘i’
  → Sorry (body): s _ _ _ i _

Hangmanの履歴依存性

- hangmanGuess ‘a’
  → Sorry (head): _ _ _ _ _ _ _
- hangmanGuess ‘i’
  → Good guess!: _ _ _ _ i _
- hangmanGuess ‘s’
  → Good guess!: s _ _ _ i _
- hangmanGuess ‘i’
  → Sorry (body): s _ _ _ i _

hangmanGuessの特徴

- 同じ回答に対して異なる反応
  - すでに判明している文字を答えても罰を被る
  - 間違えたときには常に新しい体躯の部分を表示する

機能の分析と状態

- 2つの機能
  - hangman
  - hangmanGuess
- 3つの状態
  - 進行中のゲームで親が選んだ単語 (chosenWord: List[Char])
  - 子が言い当てた文字を表現したもの (knownLetters)
  - まだ表示されてない躯体の箇所 (bodyPartsLeft)

hangmanGuess

正解	事前の知識	回答	結果・副作用
B A L L	B _ _ _	L	Good guess b _ l l
B A L L	B _ L L	A	You won!
B A L L	B _ L L	L	L-arm, R-arm, L-leg, … ⇒ R-arm, L-leg, …
B A L L	B _ L L	L	R-Leg ⇒ The End

nextKnownLetters vs knownLetters

事前の知識	回答	事後の知識	知識の追加は？	正解？
B _ _ _	L	B _ L L	true	false
B _ L L	A	B A L L	true	true
B _ L L	L	B _ L L	false	false

hangmanGuess: 非関数的振舞い

- hangman
- **hangmanGuess('a')**
  → Sorry (head): _ _ _ _ _ _ _
- hangmanGuess('i')
  → Good guess! _ _ _ _ i _
- hangmanGuess('s')
  → Good guess! s _ _ _ i _
- hangmanGuess('i')
  → Sorry (body): s _ _ _ i _
- …
- hangmanGuess('d')
  → You wan!
- hangman
- **hangmanGuess('a')**
  → Good guess! _ _ _ _ _ a m