-- テトリス123（モード選択で123に対応）

{-# LANGUAGE NamedFieldPuns, RecordWildCards #-}

{-# LANGUAGE GADTs #-}

import Data.Char (toLower, toUpper)

import Graphics.Gloss

import Graphics.Gloss.Interface.Pure.Game

import Data.List (sort)

import Data.List

import Data.Char

import System.IO

import System.Random

import qualified Data.Map as M

import Data.Map ((!))

import Debug.Trace

import Control.Applicative

import System.CPUTime

-- ゲームの状態 (点数、置いたミノ、動かすミノ)

data GameState = GameState { init\_stop :: Bool, score :: Int, clear\_line :: Int, box :: Box, randomg :: StdGen, old\_randomg :: StdGen, level :: Int, old\_level :: Int, tmp\_level :: Int, levelUp :: Bool, hold :: Int, hold\_num :: Int, mino\_list :: [Int], old\_mino\_list :: [Int], mino\_time :: Int, danger :: Int, mino\_i :: Int, tetris :: Bool, tmp\_tetris :: Int, ren :: Int, allclear :: Bool, tmp\_allclear :: Int, pair :: Int, times :: Int, auto\_fall :: Bool, select\_mino :: Bool, special :: Bool, mode\_select :: Int, now :: Int, ozyama :: [Char], kai :: Int, big :: Bool, tmp\_fixed :: Int, fixed :: Int }

-- deriving Show

-- mcolor : ミノの色, mshape : ミノの形

data Mino = Mino { mcolor :: Color, mshape :: [(Int, Int)] }

-- すでに置いたミノの配列

type Box = M.Map (Int,Int) Color

type Time = Float

emptyColor = white

seven\_minos :: [Mino]

seven\_minos = [

-- Oミノ

Mino{ mcolor = yellow,

mshape = [(2,0),(2,1),(1,0),(1,1)]},

-- Jミノ

Mino{ mcolor = blue,

mshape = [(0,0),(0,1),(1,0),(2,0)]},

-- Lミノ

Mino{ mcolor = orange,

mshape = [(0,0),(2,1),(1,0),(2,0)]},

-- Sミノ

Mino{ mcolor = green,

mshape = [(0,0),(1,0),(1,1),(2,1)]},

-- Zミノ

Mino{ mcolor = red,

mshape = [(0,1),(1,1),(1,0),(2,0)]},

-- Iミノ

Mino{ mcolor = cyan,

mshape = [(0,1),(1,1),(2,1),(3,1)]},

-- Tミノ

Mino{ mcolor = violet,

mshape = [(0,0),(1,0),(1,1),(2,0)]}

]

big\_minos :: [Mino]

big\_minos = [

-- Oミノ

Mino{ mcolor = yellow,

mshape = [(2,0),(2,1),(1,0),(1,1),(0,1),(3,1),(0,0),(0,-1),(0,-2),(1,-1),(1,-2),(2,-1),(2,-2),(3,0),(3,-1),(3,-2)]},

-- Jミノ

Mino{ mcolor = blue,

mshape = [(-1,1),(0,1),(-1,0),(0,0),(-1,-1),(0,-1),(1,-1),(2,-1),(3,-1),(4,-1),(-1,-2),(0,-2),(1,-2),(2,-2),(3,-2),(4,-2)]},

-- Lミノ

Mino{ mcolor = orange,

mshape = [(3,1),(4,1),(3,0),(4,0),(-1,-1),(0,-1),(1,-1),(2,-1),(3,-1),(4,-1),(-1,-2),(0,-2),(1,-2),(2,-2),(3,-2),(4,-2)]},

-- Sミノ

Mino{ mcolor = green,

mshape = [(-1,-2),(0,-2),(1,-2),(2,-2),(-1,-1),(0,-1),(1,-1),(2,-1),(1,0),(2,0),(1,1),(2,1),(3,1),(4,1),(3,0),(4,0)]},

-- Zミノ

Mino{ mcolor = red,

mshape = [(-1,1),(0,1),(1,1),(2,1),(-1,0),(0,0),(1,0),(2,0),(1,-1),(2,-1),(3,-1),(4,-1),(1,-2),(2,-2),(3,-2),(4,-2)]},

-- Iミノ

Mino{ mcolor = cyan,

mshape = [(-2,1),(-1,1),(0,1),(1,1),(2,1),(3,1),(4,1),(5,1),(-2,0),(-1,0),(0,0),(1,0),(2,0),(3,0),(4,0),(5,0)]},

-- Tミノ

Mino{ mcolor = violet,

mshape = [(1,0),(1,1),(2,0),(2,1),(-1,-1),(0,-1),(1,-1),(2,-1),(3,-1),(4,-1),(-1,-2),(0,-2),(1,-2),(2,-2),(3,-2),(4,-2)]}

]

-- ミノがないとき

blank\_mino :: Mino

blank\_mino = Mino { mcolor = white, mshape = [] }

light\_blue :: Color

light\_blue = makeColor 0.6 0.9 1 1

dark\_blue :: Color

dark\_blue = makeColor 0.2 0.5 0.9 1

-- タイトル画面

tytleScreen :: GameState -> Picture

tytleScreen s@GameState{..} = pictures[

translate 0 8 $ pictures[

color light\_blue $ rectangleSolid 500 500,

color dark\_blue $ polygon [(-20,85),(-60,85),(-60,125),(60,125),(60,85),(20,85),(20,45),(-20,45),(-20,85)],

color yellow $ line [(-20,85),(-60,85),(-60,125),(60,125),(60,85),(20,85),(20,45),(-20,45),(-20,85)],

drawTETRIS],

if now == 5 then pictures [ drawstopScreen s,

color white $ polygon [(-60,15),(-60,-10),(60,-10),(60,15)],

translate (-55) (-5) $ scale 0.15 0.15 $ text "Press",

translate 0 (-5) $ scale 0.15 0.15 $ text "Enter!" ]

else pictures [ translate (-55) (35) $ scale 0.08 0.08 $ text "Mode",

translate (-65) (22) $ scale 0.08 0.08 $ text "Auto Fall",

translate (-70) (9) $ scale 0.08 0.08 $ text "Select Mino",

translate (-60) (-4) $ scale 0.08 0.08 $ text "Special",

translate (-5) (22) $ scale 0.05 0.05 $ text "OFF",

translate (15) (22) $ scale 0.05 0.05 $ text "ON",

translate (-5) (9) $ scale 0.05 0.05 $ text "OFF",

translate (15) (9) $ scale 0.05 0.05 $ text "ON",

translate (-5) (-4) $ scale 0.05 0.05 $ text "OFF",

translate (15) (-4) $ scale 0.05 0.05 $ text "ON",

translate (-5) (35) $ scale 0.05 0.05 $ text "Free",

translate (15) (35) $ scale 0.05 0.05 $ text "20line",

translate (35) (35) $ scale 0.05 0.05 $ text "10000score",

translate (0) (60) $ color white $ rectangleSolid 115 15,

translate (-55) (55) $ scale 0.08 0.08 $ text "Press Enter to Start!",

translate 6 3 $ pictures [if mode\_select == 1 then translate (-5) (35) $ rectangleWire 20 10

else if mode\_select == 2 then translate (17) (35) $ rectangleWire 20 10

else translate (47) (35) $ rectangleWire 37 10,

if not auto\_fall then translate (-5) (22) $ rectangleWire 20 10

else translate (15) (22) $ rectangleWire 20 10,

if not select\_mino then translate (-5) (9) $ rectangleWire 20 10

else translate (15) (9) $ rectangleWire 20 10,

if not special then translate (-5) (-4) $ rectangleWire 20 10

else translate (15) (-4) $ rectangleWire 20 10

],

color yellow $ translate 6 3 $ pictures [

if mode\_select == 1 && now == 1 then translate (-5) (35) $ rectangleWire 20 10

else if mode\_select == 2 && now == 1 then translate (17) (35) $ rectangleWire 20 10

else if mode\_select == 3 && now == 1 then translate (47) (35) $ rectangleWire 37 10

else if not auto\_fall && now == 2 then translate (-5) (22) $ rectangleWire 20 10

else if auto\_fall && now == 2 then translate (15) (22) $ rectangleWire 20 10

else if not select\_mino && now == 3 then translate (-5) (9) $ rectangleWire 20 10

else if select\_mino && now == 3 then translate (15) (9) $ rectangleWire 20 10

else if not special && now == 4 then translate (-5) (-4) $ rectangleWire 20 10

else if special && now == 4 then translate (15) (-4) $ rectangleWire 20 10

else blank

]

]

]

drawAllclear :: Picture

drawAllclear = color yellow $ translate (-50) 70 $ scale 0.4 0.4 $ pictures[

-- A

translate 10 0 $ pictures[

polygon [(2,2),(8,2),(23,38),(17,38)],

polygon [(32,2),(38,2),(23,38),(17,38)],

polygon [(10,20),(10,14),(30,14),(30,20)]

],

translate 70 (-40) $ pictures[

polygon [(2,2),(8,2),(23,38),(17,38)],

polygon [(32,2),(38,2),(23,38),(17,38)],

polygon [(10,20),(10,14),(30,14),(30,20)]

],

-- L

translate 50 0 $ polygon [(8,8),(28,8),(28,2),(2,2),(2,38),(8,38)],

translate 80 0 $ polygon [(8,8),(28,8),(28,2),(2,2),(2,38),(8,38)],

translate 10 (-40) $ polygon [(8,8),(28,8),(28,2),(2,2),(2,38),(8,38)],

-- C

translate (-30) (-40) $ polygon [(26,32),(32,26),(36,30),(28,38),(12,38),(14,32)],

translate (-30) (-40) $ polygon [(8,26),(14,32),(12,38),(2,28),(2,12),(8,14)],

translate (-30) (-40) $ polygon [(14,8),(8,14),(2,12),(12,2),(28,2),(26,8)],

translate (-30) (-40) $ polygon [(26,8),(28,2),(36,10),(32,14)],

-- E

translate 40 (-40) $ polygon [(8,32),(28,32),(28,38),(2,38),(2,2),(8,2)],

translate 40 (-40) $ polygon [(8,23),(8,17),(28,17),(28,23)],

translate 40 (-40) $ polygon [(8,2),(28,2),(28,8),(2,8)],

-- R

translate 110 (-40) $ pictures [

polygon [(8,32),(28,32),(28,38),(2,38),(2,2),(8,2)],

polygon [(22,26),(2,26),(2,20),(28,20),(28,38),(22,38)],

polygon [(12,20),(21,2),(28,2),(19,20)]

]

]

drawTETRIS :: Picture

drawTETRIS = pictures[

translate (-60) 85 $ color red $ polygon [(7,32),(7,2),(13,2),(13,32),(18,32),(18,38),(2,38),(2,32)],

translate (-40) 85 $ color orange $ polygon [(8,8),(8,38),(2,38),(2,2),(18,2),(18,8),(8,8)],

translate (-40) 85 $ color orange $ polygon [(2,38),(18,38),(18,32),(2,32)],

translate (-40) 85 $ color orange $ polygon [(2,23),(18,23),(18,17),(2,17)],

translate (-20) 85 $ color yellow $ polygon [(7,32),(7,2),(13,2),(13,32),(18,32),(18,38),(2,38),(2,32)],

translate 20 85 $ color cyan $ polygon [(7,32),(13,32),(13,38),(7,38)],

translate 20 85 $ color cyan $ polygon [(7,2),(13,2),(13,26),(7,26)],

translate 40 85 $ color violet $ polygon [(8,32),(18,8),(12,8),(2,32),(2,38),(18,38),(18,32)],

translate 40 85 $ color violet $ polygon [(2,2),(2,8),(18,8),(18,2)],

translate 0 85 $ color green $ polygon [(12,32),(2,32),(2,38),(18,38),(18,20),(12,20)],

translate 0 85 $ color green $ polygon [(2,38),(8,38),(8,2),(2,2)],

translate 0 85 $ color green $ polygon [(8,20),(13,2),(18,2),(13,20)],

translate 0 85 $ color green $ polygon [(2,26),(18,26),(18,20),(2,20)]

]

drawTETRIS\_black :: Picture

drawTETRIS\_black = pictures[

translate (-60) 85 $ color gray $ polygon [(7,32),(7,2),(13,2),(13,32),(18,32),(18,38),(2,38),(2,32)],

translate (-40) 85 $ color gray $ polygon [(8,8),(8,38),(2,38),(2,2),(18,2),(18,8),(8,8)],

translate (-40) 85 $ color gray $ polygon [(2,38),(18,38),(18,32),(2,32)],

translate (-40) 85 $ color gray $ polygon [(2,23),(18,23),(18,17),(2,17)],

translate (-20) 85 $ color gray $ polygon [(7,32),(7,2),(13,2),(13,32),(18,32),(18,38),(2,38),(2,32)],

translate 20 85 $ color gray $ polygon [(7,32),(13,32),(13,38),(7,38)],

translate 20 85 $ color gray $ polygon [(7,2),(13,2),(13,26),(7,26)],

translate 40 85 $ color gray $ polygon [(8,32),(18,8),(12,8),(2,32),(2,38),(18,38),(18,32)],

translate 40 85 $ color gray $ polygon [(2,2),(2,8),(18,8),(18,2)],

translate 0 85 $ color gray $ polygon [(12,32),(2,32),(2,38),(18,38),(18,20),(12,20)],

translate 0 85 $ color gray $ polygon [(2,38),(8,38),(8,2),(2,2)],

translate 0 85 $ color gray $ polygon [(8,20),(12,2),(18,2),(12,20)],

translate 0 85 $ color gray $ polygon [(2,26),(18,26),(18,20),(2,20)]

]

backgroundColor :: Color

backgroundColor = makeColor 0.3 0.5 0.7 1

main :: IO ()

main = do

--n <- randomRIO (0,100000)

play (InWindow "Tetris 1" (600,600) (20,20))

-- mkStdGen n として上のコメント化を解除すれば,ランダムになる

black 30 (start (mkStdGen 1))

drawWorld

eventHandler

frameHandler

start :: StdGen -> Freer ()

start gen = mainloop initState

where

initState = GameState { init\_stop = True, score = 0, clear\_line = 0, randomg = gen, old\_randomg = gen, box = emptyBox, level = 0, old\_level = 1, tmp\_level = 0, levelUp = False, hold = -1, hold\_num = 0, mino\_list = [0..6], old\_mino\_list = [0..6], mino\_time = 0 , danger = 0, mino\_i = -1, tetris = False, tmp\_tetris = 0, ren = 0, allclear = False, tmp\_allclear = 0, pair = 0, times = 0, mode\_select = 1, auto\_fall = False, select\_mino = False, special = False, now = 1, ozyama = [], kai = 0, big = False, tmp\_fixed = 0, fixed = -1 }

-- ここにメインの挙動を記述する。ここから手をつけていく

mainloop :: GameState -> Freer ()

-- タイトル画面の表示

mainloop s@GameState{..} | init\_stop = do

key <- getKey s (pictures[ tytleScreen s ])

let s' = if key == Just "right" && now == 1 && mode\_select == 1 then s{ mode\_select = 2 }

else if key == Just "left" && now == 1 && mode\_select == 2 then s{ mode\_select = 1 }

else if key == Just "right" && now == 1 && mode\_select == 2 then s{ mode\_select = 3 }

else if key == Just "left" && now == 1 && mode\_select == 3 then s{ mode\_select = 2 }

else if key == Just "down" && now == 1 then s{ now = 2 }

else if key == Just "up" && now == 2 then s{ now = 1 }

else if key == Just "right" && now == 2 && not auto\_fall then s{ auto\_fall = True }

else if key == Just "left" && now == 2 && auto\_fall then s{ auto\_fall = False }

else if key == Just "down" && now == 2 then s{ now = 3 }

else if key == Just "up" && now == 3 then s{ now = 2 }

else if key == Just "right" && now == 3 && not select\_mino then s{ select\_mino = True }

else if key == Just "left" && now == 3 && select\_mino then s{ select\_mino = False }

else if key == Just "down" && now == 3 then s{ now = 4 }

else if key == Just "up" && now == 4 then s{ now = 3 }

else if key == Just "right" && now == 4 && not special then s{ special = True }

else if key == Just "left" && now == 4 && special then s{ special = False }

else if key == Just "down" && now == 4 then s{ now = 5 }

else if key == Just "up" && now == 5 then s{ now = 4 }

else if key == Just "drop" && now == 5 then s{ init\_stop = False }

else if key == Just "drop" then s{ now = 5 }

-- 隠しコマンド。bを押すとミノが大きくなる

else if key == Just "b" then s{ big = True }

else s

mainloop s'

-- 複数ラインを一気に消す。画面の表示も

mainloop s@GameState{..} | length (findFullLines box) > 0 = do

let fullLines = findFullLines box

n = length fullLines

score\_now = if n == 1 then score + 100 \* (ren+1)

else if n == 2 then score + 300 \* (ren+1)

else if n == 3 then score + 500 \* (ren+1)

else if n == 4 then score + 1000 \* (ren+1)

else if n == 5 then score + 1500 \* (ren+1)

else if n == 6 then score + 2000 \* (ren+1)

else if n == 7 then score + 2500 \* (ren+1)

else if n == 8 then score + 3000 \* (ren+1)

else score

level\_now = (score\_now `div` 500)

s' = s{ clear\_line = clear\_line + n, pair = pairJudge fullLines, box = collapseLines fullLines box, ren = ren + 1,

level = if level\_now <= 20 then level\_now else 20,

--levelUp = if level /= old\_level then True else False,

tmp\_level = if level\_now /= old\_level then 1 else 0,

tetris = if n >= 4 || tetris then True else False,

score = score\_now,

tmp\_tetris = if n >= 4 then 1 else 0

}

pair' = pairJudge fullLines

--getKey s' (pictures[

pause s' (pictures[

if mode\_select == 1 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "Free Play"

else if mode\_select == 2 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "20 line"

else color white $ translate (-75) (125) $ scale 0.04 0.04 $ text "10000 score",

if big then color white $ translate (-75) (115) $ scale 0.05 0.05 $ text "Big Mode" else blank,

color white $ translate 20 100 $ scale 0.05 0.05 $ text ("SCORE " ++ show score),

color white $ translate 20 90 $ scale 0.05 0.05 $ text ("LINES " ++ show clear\_line),

color white $ translate 22 70 $ scale 0.05 0.05 $ text "HOLD ",

if level <= 19 then color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL " ++ show level)

else color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL MAX"),

if tmp\_tetris > 0 then color yellow $ translate 15 20 $ scale 0.1 0.1 $ text "tetris!!" else blank,

if tetris then translate 0 45 $ scale 0.7 0.7 $ drawTETRIS else blank,

if ren > 1 then color yellow $ translate 15 10 $ scale 0.1 0.1 $ text ("ren " ++ show (ren-1)) else blank,

--if tmp\_tetris > 0 then color white $ translate 15 20 $ scale 0.1 0.1 $ text "tetris!!" else blank,

--color white $ translate 15 20 $ scale 0.1 0.1 $ text ("" ++ show fullLines),

--color white $ translate 15 10 $ scale 0.1 0.1 $ text ("" ++ show pair'),

color white $ line [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

if hold == -1 then drawHoldMino blank\_mino s

else drawHoldMino (seven\_minos !! hold) s,

if tmp\_allclear > 0 then drawAllclear else blank,

if tmp\_level > 0 then color yellow $ translate 25 76 $ scale 0.03 0.03 $ text "Lv UP!" else blank,

color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text ozyama,

if tmp\_fixed > 0 then color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text "fixed" else blank

])

let s'' = if isAllBlank s' then s'{ allclear = True, tmp\_allclear = 1, score = score\_now + 2000 }

else s'{ allclear = False, box = if pair' == 1 then copyLines fullLines s' [fullLines !! 0]

else if pair' == 2 then copyLines fullLines s' [fullLines !! 0]

else if pair' == 3 then copyLines fullLines s' [fullLines !! 0]

else if pair' == 4 then copyLines fullLines s' [fullLines !! 0, (fullLines !! 0)+1]

else copyLines fullLines s' []

}

mainloop s''

-- 上に積み上がったのでゲームを終わりにする

mainloop s | isFinished s = finish s

-- ゲームをクリアしたら終わり

mainloop s | isCleared s = finish s

-- テトリス2ではここのミノを落とした後ここでループさせる

mainloop s@GameState{..} | select\_mino = do

key <- getKey s (pictures[

if mode\_select == 1 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "Free Play"

else if mode\_select == 2 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "20 line"

else color white $ translate (-75) (125) $ scale 0.04 0.04 $ text "10000 score",

if big then color white $ translate (-75) (115) $ scale 0.05 0.05 $ text "Big Mode" else blank,

color white $ translate 20 100 $ scale 0.05 0.05 $ text ("SCORE " ++ show score),

color white $ translate 20 90 $ scale 0.05 0.05 $ text ("LINES " ++ show clear\_line),

color white $ translate (-40) (-10) $ scale 0.05 0.05 $ text ("PUSH KEY BOAD"),

color white $ translate 22 70 $ scale 0.05 0.05 $ text "HOLD ",

--color white $ translate 22 110 $ scale 0.05 0.05 $ text ("tmp\_tetris " ++ show tmp\_tetris),

if level <= 19 then color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL " ++ show level)

else color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL MAX"),

if tmp\_tetris > 0 then color yellow $ translate 15 20 $ scale 0.1 0.1 $ text "tetris!!" else blank,

if ren > 1 then color yellow $ translate 15 10 $ scale 0.1 0.1 $ text ("ren " ++ show (ren-1)) else blank,

--color white $ translate 15 110 $ scale 0.1 0.1 $ text ("ren " ++ show (ren-1)),

--color white $ translate 0 110 $ scale 0.05 0.05 $ text ("list " ++ show mino\_list),

--color white $ translate 0 120 $ scale 0.05 0.05 $ text ("mino " ++ show mino\_i),

color white $ line [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

if tetris then translate 0 45 $ scale 0.7 0.7 $ drawTETRIS else blank,

if tmp\_allclear > 0 then drawAllclear else blank,

if tmp\_level > 0 then color yellow $ translate 25 76 $ scale 0.03 0.03 $ text "Lv UP!" else blank,

color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text ozyama,

if tmp\_fixed > 0 then color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text "fixed" else blank

])

let mino = if key == Just "o" then 0

else if key == Just "j" then 1

else if key == Just "l" then 2

else if key == Just "s" then 3

else if key == Just "z" then 4

else if key == Just "i" then 5

else if key == Just "t" then 6

else -1

level\_now = (score `div` 500)

s' = s{ level = if level\_now <= 20 then level\_now else 20 }

if mino /= -1 then lowerMino (3,18) (seven\_minos !! mino) s'

else if key == Just "r" then start randomg

-- テトリス2でミノ待ち画面時のポーズに対応

else if key == Just "p" then stopScreen2 s

else mainloop s'

-- 最初またはミノを落とした後にミノを用意する

mainloop s@GameState{..} = do

let (mino, randomg') = if tmp\_fixed > 0 then (seven\_minos !! fixed, randomg)

else chooseMino s randomg

s' = s{ old\_randomg = randomg, randomg = randomg', hold\_num = 0, old\_mino\_list = mino\_list, mino\_list = a, mino\_time = 0, times = 0, kai = kai + 1,

mino\_i = if length mino\_list == 0 then (fst (randomR (0,6) randomg))

else mino\_list !! (fst (randomR (0, length mino\_list-1) randomg))

}

if b == 5 then lowerMino (3,18) mino s'

else if isHalfFull s then lowerMino (3,19) mino s'

else lowerMino (3,18) mino s'

where

a = if length mino\_list == 0 then delete (fst (randomR (0, 6) randomg)) [0..6]

else delete (mino\_list !! (fst (randomR (0, length mino\_list-1) randomg))) mino\_list

b = if length mino\_list == 0 then (fst (randomR (0,6) randomg))

else mino\_list !! (fst (randomR (0, length mino\_list-1) randomg))

-- ラインを消すときに途中が空いている特別なパターンを判別する

pairJudge :: [Int] -> Int

pairJudge lines =

if length lines == 3 && lines !! 2 - lines !! 1 == 2 then 1

else if length lines == 3 && lines !! 1 - lines !! 0 == 2 then 2

else if length lines == 2 && lines !! 1 - lines !! 0 == 2 then 3

else if length lines == 2 && lines !! 1 - lines !! 0 == 3 then 4

else 0

-- ゲームが終わった後に表示する。ループさせて表示し続ける

finish :: GameState -> Freer()

finish s@GameState{..} = do

key <- getKey s (pictures[

color white $ translate 20 100 $ scale 0.05 0.05 $ text ("score " ++ show score),

color white $ translate 20 90 $ scale 0.05 0.05 $ text ("lines " ++ show clear\_line),

color white $ translate 22 70 $ scale 0.05 0.05 $ text "hold ",

color white $ line [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

if level <= 19 then color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL " ++ show level)

else color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL MAX"),

if hold == -1 then drawHoldMino blank\_mino s

else drawHoldMino (seven\_minos !! hold) s,

color (makeColor 0 0 0 0.7) $ polygon [(-1000,-1000),(-1000,1000),(1000,1000),(1000,-1000)],

if isCleared s then color white $ translate (-40) 70 $ scale 0.1 0.1 $ text "You Win!!"

else color white $ translate (-40) 70 $ scale 0.1 0.1 $ text "You Lose...",

color white $ translate (-40) 40 $ scale 0.1 0.1 $ text "Retry: R key"

])

-- rキーが押されたらリトライ

if key == Just "r" then start randomg

else finish s

drawstopScreen :: GameState -> Picture

drawstopScreen GameState{..} = pictures [translate 0 20 $ scale 0.9 0.9 $ pictures[

color white $ translate 20 100 $ scale 0.05 0.05 $ text ("score " ++ show score),

color white $ translate 20 90 $ scale 0.05 0.05 $ text ("lines " ++ show clear\_line),

if level <= 19 then color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL " ++ show level)

else color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL MAX"),

color white $ translate 22 70 $ scale 0.05 0.05 $ text "hold ",

color white $ line [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

color (makeColor 0 0 0 0.7) $ polygon [(-1000,-1000),(-1000,1000),(1000,1000),(1000,-1000)],

color (makeColor 0.9 0.9 0.9 1) $ polygon [(-60,0),(-60,120),(55,120),(55,0)],

color green $ line [(-60,0),(-60,120),(55,120),(55,0),(-60,0)],

translate (-55) 105 $ scale 0.09 0.09 $ text "Manual",

translate (-45) 90 $ scale 0.05 0.05 $ text "Hard Drop",

translate (-10) 90 $ scale 0.05 0.05 $ text ": Enter ",

translate (-45) 80 $ scale 0.05 0.05 $ text "CW Spin",

translate (-10) 80 $ scale 0.05 0.05 $ text ": S or Space",

translate (-45) 70 $ scale 0.05 0.05 $ text "UCW Spin",

translate (-10) 70 $ scale 0.05 0.05 $ text ": A",

translate (-45) 60 $ scale 0.05 0.05 $ text "Retry",

translate (-10) 60 $ scale 0.05 0.05 $ text ": R",

translate (-45) 50 $ scale 0.05 0.05 $ text "Pause",

translate (-10) 50 $ scale 0.05 0.05 $ text ": P",

if not select\_mino then translate (-45) 40 $ scale 0.05 0.05 $ text "Hold" else blank,

if not select\_mino then translate (-10) 40 $ scale 0.05 0.05 $ text ": H" else blank,

if select\_mino then translate (-45) 30 $ scale 0.05 0.05 $ text "Mino" else blank,

if select\_mino then translate (-10) 30 $ scale 0.05 0.05 $ text ": O J L S Z I T" else blank,

translate (-45) 15 $ scale 0.05 0.05 $ text "Move",

translate (-10) 15 $ scale 0.03 0.03 $ text ": - -",

if not auto\_fall then translate (3) 20 $ scale 0.02 0.02 $ text "|" else blank,

translate (3) 10 $ scale 0.02 0.02 $ text "|",

if not auto\_fall then polygon [(1.2,21),(3.2,23),(5.2,21)] else blank,

polygon [(1.2,11),(3.2,9),(5.2,11)],

polygon [(-3.5,16.4),(-1.5,14.4),(-1.5,18.4)],

polygon [(10.2,16.4),(8.2,14.4),(8.2,18.4)]],

color white $ polygon [(-60,15),(-60,-10),(60,-10),(60,15)],

translate (-55) (-5) $ scale 0.15 0.15 $ text "Press",

translate 0 (-5) $ scale 0.15 0.15 $ text "Enter!"

]

stopScreen :: (Int,Int) -> Mino -> GameState -> Freer()

stopScreen pt mino s@GameState{..} = do

key <- getKey s (drawstopScreen s)

-- rキーが押されたらリトライ

if key == Just "r" then start randomg

else if key == Just "p" || key == Just "drop" then lowerMino pt mino s

else stopScreen pt mino s

stopScreen2 :: GameState -> Freer()

stopScreen2 s@GameState{..} = do

key <- getKey s (drawstopScreen s)

-- rキーが押されたらリトライ

if key == Just "r" then start randomg

else if key == Just "p" || key == Just "drop" then mainloop s

else stopScreen2 s

-- 一番上から2つめの行が埋まっているかを判定

isHalfFull :: GameState -> Bool

isHalfFull GameState{..} = any (\i -> box ! (i,18) /= emptyColor) [3..6]

-- ミノが一つもないかを判断

isAllBlank :: GameState -> Bool

isAllBlank GameState{..} = all (== emptyColor) box

-- ゲームを終わらせるか判定する

isFinished :: GameState -> Bool

isFinished GameState{..} = any (\i -> box ! (i,19) /= emptyColor) [3..6]

-- ゲームをクリアしたかどうかを判定する

isCleared :: GameState -> Bool

isCleared GameState{..} = if clear\_line >= 20 && mode\_select == 2 then True

else if score >= 10000 && mode\_select == 3 then True

else False

-- 7つのミノからどれを落とすか選ぶ

chooseMino :: GameState -> StdGen -> (Mino,StdGen)

chooseMino s@GameState{..} gen =

let (a,b) = if length mino\_list == 0 then randomR (0,6) gen

else (mino\_list !! fst (randomR (0, length mino\_list-1) gen), snd (randomR (0, length mino\_list-1) gen))

in if big then (big\_minos !! a, b)

else if ozyama /= "big" then (seven\_minos !! a, b)

else (big\_minos !! a, b)

-- 一瞬でミノを下に落とす

lowerPtAsPossible :: (Int,Int) -> Mino -> GameState -> (Int,Int)

lowerPtAsPossible pt mino s =

if not (canLower pt mino s) then pt

else lowerPtAsPossible (lowerPt pt) mino s

timeover :: GameState -> Bool

timeover s@GameState{..} = if mino\_time <= 15 then True else False

-- ミノを下に落とす。ここで画面の表示も行う

lowerMino :: (Int,Int) -> Mino -> GameState -> Freer()

lowerMino pt mino s@GameState{..} | canLower pt mino s || timeover s = do

key <- getKey s (pictures[

if ozyama == "black" then polygon [(-2.4,0.3),(-52.3,0.3),(-52.3,100),(-2.4,100)] else blank,

drawMino pt mino s,

color white $ translate 20 100 $ scale 0.05 0.05 $ text ("SCORE " ++ show score),

color white $ translate 20 90 $ scale 0.05 0.05 $ text ("LINES " ++ show clear\_line),

color white $ translate 22 70 $ scale 0.05 0.05 $ text "HOLD ",

if level <= 19 then color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL " ++ show level)

else color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL MAX"),

--color white $ translate 22 40 $ scale 0.05 0.05 $ text ("tmp\_level " ++ show tmp\_level),

--color white $ translate 22 110 $ scale 0.05 0.05 $ text ("tmp\_tetris " ++ show tmp\_tetris),

--color white $ translate 22 40 $ scale 0.05 0.05 $ text ("tmp\_fixed " ++ show tmp\_fixed),

if tmp\_tetris > 0 then color yellow $ translate 15 20 $ scale 0.1 0.1 $ text "tetris!!" else blank,

if ren > 1 then color yellow $ translate 15 10 $ scale 0.1 0.1 $ text ("ren " ++ show (ren-1)) else blank,

--color white $ translate 15 110 $ scale 0.1 0.1 $ text ("ren " ++ show (ren-1)),

--color white $ translate 0 110 $ scale 0.05 0.05 $ text ("list " ++ show mino\_list),

--color white $ translate 0 120 $ scale 0.05 0.05 $ text ("times " ++ show times),

color white $ line [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

if tetris then translate 0 45 $ scale 0.7 0.7 $ drawTETRIS else blank,

if tmp\_level > 0 then color yellow $ translate 25 76 $ scale 0.03 0.03 $ text "Lv UP!" else blank,

if tmp\_allclear > 0 then drawAllclear else blank,

color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text ozyama,

if tmp\_fixed > 0 then color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text "fixed" else blank

])

let (pt', mino'') =

if key == Just "left" && okPlace (point\_add (-1,0) pt) mino s then (point\_add (-1,0) pt, mino)

else if key == Just "right" && okPlace (point\_add (1,0) pt) mino s then (point\_add (1,0) pt, mino)

else if key == Just "up" && okPlace (point\_add (0,1) pt) mino s && not auto\_fall then (point\_add (0,1) pt, mino)

else if key == Just "down" && okPlace (point\_add (0,-1) pt) mino s then (point\_add (0,-1) pt, mino)

else if (key == Just "spin" || key == Just "s") && okPlace pt (rotateCW mino) s then (pt, rotateCW mino)

else if key == Just "a" && okPlace pt (rotateUCW mino) s then (pt, rotateUCW mino)

else if key == Just "h" && hold\_num == 0 && not select\_mino then holdMino s

else if key == Just "drop" then (lowerPtAsPossible pt mino s, mino)

-- ここで pt を lowerPt pt にしたら、時間で落ちるようになる。今回は未実装

else if canLower pt mino s && auto\_fall && times == 21 - level then (lowerPt pt, mino)

-- 下のコメントを外すとテトリス2でも落下する

--else if canLower pt mino s && mode\_select == 2 && times == 21 - level then (lowerPt pt, mino)

else (pt, mino)

let s' = if key == Just "h" && hold\_num == 0 && length mino\_list == 0 && not select\_mino

then s{ mino\_list = a, hold\_num = 1, mino\_time = 0, danger = 1,

hold = if tmp\_fixed > 0 then fixed else mino\_i }

else if key == Just "h" && hold\_num == 0 && length mino\_list == 6 && not select\_mino

then s{ mino\_list = a, hold\_num = 1, mino\_time = 0, danger = 1,

mino\_i = mino\_list !! fst (randomR (0, length mino\_list-1) old\_randomg),

hold = if tmp\_fixed > 0 then fixed else mino\_i }

else if key == Just "h" && hold\_num == 0 && not select\_mino

then s{ mino\_i = if hold /= -1 then hold else mino\_list !! (fst (randomR (0, length mino\_list-1) randomg)),

mino\_list = a, hold\_num = 1, mino\_time = 0, danger = 0,

hold = if tmp\_fixed > 0 then fixed else b !! fst (randomR (0, length mino\_list) old\_randomg) }

else if (key == Just "down" && mino\_time >= 1) || key == Just "drop"

then s{ mino\_time = 15 }

else if key == Just "left" || key == Just "right" || key == Just "up" || key == Just "spin" || key == Just "s" || key == Just "a"

then s{ mino\_time = 0 }

else s{ mino\_time = if canLower pt mino s then mino\_time else mino\_time + 1 }

let s'' = s'{ times = if times < 21 - level then times + 1 else 0,

tmp\_tetris = if 0 < tmp\_tetris && tmp\_tetris < 30 then tmp\_tetris + 1 else 0,

tmp\_allclear = if 0 < tmp\_allclear && tmp\_allclear < 30 then tmp\_allclear + 1 else 0,

tmp\_level = if 0 < tmp\_level && tmp\_level < 30 then tmp\_level + 1 else 0 }

-- rキーが押されたらリトライ

if key == Just "r" then start randomg

else if key == Just "p" then stopScreen pt mino s'

else lowerMino pt' mino'' s''

where

a = if length mino\_list == 0 then delete (fst (randomR (0, 6) randomg)) [0..6]

else if hold == -1 then delete (mino\_list !! (fst (randomR (0, length mino\_list-1) randomg))) mino\_list

else mino\_list

b = if length mino\_list == 6 then mino\_list

else old\_mino\_list

-- ミノが落下し終わったら、落としたミノをMapにいれ、もう一回mainloopを繰り返す

lowerMino pt Mino{..} s@GameState{..} = do

let box' = foldr (\mp -> M.insert (point\_add\_clap mp pt) mcolor)

box mshape

level\_now = (score `div` 500)

-- おじゃまを選ぶサイコロ

dice = if kai `mod` 5 == 0 && kai /= 0 && special then [0..4] !! ((times - score) `mod` 5) else -1

lines = [1..4] !! ((times + score) `mod` 4)

empty = [0..9] !! ((times - score) `mod` 10)

s' = s{ score = score + 4, level = if level\_now <= 20 then level\_now else 20,

ren = if length (findFullLines box') == 0 then 0 else ren,

ozyama = if dice == 0 then "nothing"

else if dice == 1 then "fixed"

else if dice == 2 then "upLine"

else if dice == 3 then "big"

else if dice == 4 then "black"

else [],

tmp\_fixed = if dice == 1 then 4

else if tmp\_fixed > 0 then tmp\_fixed - 1

else 0,

fixed = if dice == 1 then [0..6] !! ((times + score) `mod` 4)

else if tmp\_fixed > 0 then fixed

else -1,

box = if dice == 2 then upLines lines empty box' else box'

}

-- 一瞬画面を止める

pause s' (pictures[

--getKey s' (pictures[

if mode\_select == 1 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "Free Play"

else if mode\_select == 2 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "20 line"

else color white $ translate (-75) (125) $ scale 0.04 0.04 $ text "10000 score",

if big then color white $ translate (-75) (115) $ scale 0.05 0.05 $ text "Big Mode" else blank,

color white $ translate 20 100 $ scale 0.05 0.05 $ text ("SCORE " ++ show score),

color white $ translate 20 90 $ scale 0.05 0.05 $ text ("LINES " ++ show clear\_line),

color white $ translate 22 70 $ scale 0.05 0.05 $ text "HOLD ",

if level <= 19 then color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL " ++ show level)

else color white $ translate 20 80 $ scale 0.05 0.05 $ text ("LEVEL MAX"),

--color white $ translate 0 110 $ scale 0.05 0.05 $ text ("list " ++ show mino\_list),

--color white $ translate 0 120 $ scale 0.05 0.05 $ text ("times " ++ show times),

if ren > 1 then color yellow $ translate 15 10 $ scale 0.1 0.1 $ text ("ren " ++ show (ren-1)) else blank,

if tmp\_tetris > 0 then color yellow $ translate 15 20 $ scale 0.1 0.1 $ text "tetris!!" else blank,

if tmp\_level > 0 then color yellow $ translate 25 76 $ scale 0.03 0.03 $ text "Lv UP!" else blank,

if tetris then translate 0 45 $ scale 0.7 0.7 $ drawTETRIS else blank,

color white $ line [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

if hold == -1 then drawHoldMino blank\_mino s

else drawHoldMino (seven\_minos !! hold) s,

if tmp\_allclear > 0 then drawAllclear else blank,

color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text ozyama,

if tmp\_fixed > 0 then color yellow $ translate (-70) 110 $ scale 0.1 0.1 $ text "fixed" else blank

])

mainloop s'

-- ホールドした後の動き

holdMino :: GameState -> ((Int,Int), Mino)

holdMino s@GameState{..} =

if hold == -1 && tmp\_fixed > 0 && big then ((3,18), big\_minos !! fixed)

else if hold == -1 && tmp\_fixed > 0 then ((3,18), seven\_minos !! fixed)

else if hold == -1 then ((3,18), fst (chooseMino s randomg))

else if danger == 1 && ozyama == "big" then ((3,18), big\_minos !! hold)

else if danger == 1 && big then ((3,18), big\_minos !! hold)

else if danger == 1 then ((3,18), seven\_minos !! hold)

else if ozyama == "big" || big then ((3,18), big\_minos !! hold)

else ((3,18), seven\_minos !! hold)

-- ミノの4つのセルの場所を記憶させ直す

point\_add\_clap :: (Int,Int) -> (Int,Int) -> (Int,Int)

point\_add\_clap (i1,j1) (i2,j2) =

(clap 0 9 (i1+i2), clap 0 19 (j1+j2))

where

clap min max n

| n < min = min

| n > max = max

| otherwise = n

-- ミノが下に行けるかの判断をする

canLower :: (Int,Int) -> Mino -> GameState -> Bool

canLower pt mino s = okPlace (lowerPt pt) mino s

-- ミノを1つ下に落とす

lowerPt :: (Int,Int) -> (Int,Int)

lowerPt (i,j) = (i,j-1)

-- ミノが次の場所に行けるかどうか判断する

okPlace :: (Int,Int) -> Mino -> GameState -> Bool

okPlace pt Mino{mshape} GameState{box} =

all(\mpt -> let p@(i,j) = point\_add mpt pt

in (j == 20 && i >= 0 && i <= 9) || (okPoint p && box ! p == emptyColor)) mshape

-- ミノが壁にぶつかっていないかを判断する

okPoint :: (Int,Int) -> Bool

okPoint (i,j) = and [i >= 0, i <= 9, j >= 0, j <= 20]

-- 座標の和をとる

point\_add :: (Int,Int) -> (Int,Int) -> (Int,Int)

point\_add (i1, j1) (i2, j2) = (i1+i2, j1+j2)

-- ホールドしているミノを描く

drawHoldMino :: Mino -> GameState -> Picture

drawHoldMino Mino{..} GameState{..} = uncurry translate (boxToWorldRel (a, 10)) $ translate b 0 $ color mcolor $ shapes

where

shapes = pictures $ map drawCell mshape

a = if hold == 5 then 14

else if hold == 0 then 14

else 15

b = if hold == 0 then 3

else if hold == 5 then 2.5

else 0

-- ミノをGUIに描く。マス目をミノの前に書きたいからここでマス目を表示させる

drawMino :: (Int,Int) -> Mino -> GameState -> Picture

drawMino pt Mino{..} s@GameState{..} = pictures[

if mode\_select == 1 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "Free Play"

else if mode\_select == 2 then color white $ translate (-75) (125) $ scale 0.05 0.05 $ text "20 line"

else color white $ translate (-75) (125) $ scale 0.04 0.04 $ text "10000 score",

if big then color white $ translate (-75) (115) $ scale 0.05 0.05 $ text "Big Mode" else blank,

if hold == -1 then drawHoldMino blank\_mino s

else if danger == 1 then drawHoldMino (seven\_minos !! hold) s

else drawHoldMino (seven\_minos !! hold) s,

color white $ line [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

uncurry translate (boxToWorldRel pt) $ color mcolor $ shapes,

pictures [translate (5\*x) 0 $ color gray $

line [(fst ul - boxWidth\_half, snd ul + boxWidth\_half),

(fst bl - boxWidth\_half, snd bl - boxWidth\_half)] | x <- [1..9]],

pictures [translate 0 (5\*x) $ color gray $

line [(fst br + boxWidth\_half, snd br - boxWidth\_half),

(fst bl - boxWidth\_half, snd bl - boxWidth\_half)] | x <- [1..20]]

]

where

shapes = pictures $ map drawCell mshape

ul = boxToWorld (0,19)

ur = boxToWorld (9,19)

bl = boxToWorld (0,0)

br = boxToWorld (9,0)

-- ミノの4つそれぞれのブロックの位置

drawCell :: (Int,Int) -> Picture

drawCell pt = pictures[ polygon [(x-boxWidth\_half, y-boxWidth\_half),

(x-boxWidth\_half, y+boxWidth\_half),

(x+boxWidth\_half, y+boxWidth\_half),

(x+boxWidth\_half, y-boxWidth\_half)],

translate (-52.5) 0 $ uncurry translate (boxToWorldRel pt) $ scale 0.1777 0.1777 $

pictures [color (makeColor 0.1 0.1 0.1 0.1) $ polygon [(0,0),(30,0),(25,5),(5,5)]],

translate (-52.5) 0 $ uncurry translate (boxToWorldRel pt) $ scale 0.1777 0.1777 $

pictures [color (makeColor 0.2 0.2 0.2 0.3) $ polygon [(0,0),(5,5),(5,25),(0,30)]],

translate (-52.5) 0 $ uncurry translate (boxToWorldRel pt) $ scale 0.1777 0.1777 $

pictures [color (makeColor 0.8 0.8 0.8 0.5) $ polygon [(30,0),(25,5),(25,25),(30,30)]],

translate (-52.5) 0 $ uncurry translate (boxToWorldRel pt) $ scale 0.1777 0.1777 $

pictures [color (makeColor 0.9 0.9 0.9 0.5) $ polygon [(0,30),(30,30),(25,25),(5,25)]],

translate (-52.5) 0 $ uncurry translate (boxToWorldRel pt) $ scale 0.1777 0.1777 $

pictures [color (makeColor 0.9 0.9 0.9 0.8) $ polygon [(5,25),(25,25),(5,20)]],

translate (-52.5) 0 $ uncurry translate (boxToWorldRel pt) $ scale 0.1777 0.1777 $

pictures [color (makeColor 0.9 0.9 0.9 0.5) $ polygon [(5,15),(5,20),(25,25),(25,20)]]

]

where (x,y) = boxToWorld pt

-- ミノの形と色を出力する

drawBox :: Box -> Picture

drawBox box =

pictures $ M.foldlWithKey' (\ps pt c ->

if c == emptyColor then ps

else (color c $ drawCell pt):ps

--else ps

) [drawEmptyBox] box

-- 常に表示されるところ、枠やマス目など

drawEmptyBox :: Picture

drawEmptyBox = pictures[

color backgroundColor $ polygon [(-52.5,-100),(-52.5,300),(-1000,-100),(-1000,300)],

color backgroundColor $ polygon [(-100,0),(-100,-100),(1000,-100),(1000,0)],

color backgroundColor $ polygon [(-2.5,-100),(-2.5,300),(200,300),(200,-100)],

color backgroundColor $ polygon [(-100,100),(-100,300),(300,300),(300,100)],

color black $ polygon [(18,72),(15,72),(15,40),(45,40),(45,72),(35,72)],

translate 0 45 $ scale 0.7 0.7 $ drawTETRIS\_black,

color white $ line [(fst ul - boxWidth\_half, snd ul + boxWidth\_half),

(fst bl - boxWidth\_half, snd bl - boxWidth\_half),

(fst br + boxWidth\_half, snd br - boxWidth\_half),

(fst ur + boxWidth\_half, snd ur + boxWidth\_half)],

pictures [translate (5\*x) 0 $ color gray $

line [(fst ul - boxWidth\_half, snd ul + boxWidth\_half),

(fst bl - boxWidth\_half, snd bl - boxWidth\_half)] | x <- [1..9]],

pictures [translate 0 (5\*x) $ color gray $

line [(fst br + boxWidth\_half, snd br - boxWidth\_half),

(fst bl - boxWidth\_half, snd bl - boxWidth\_half)] | x <- [1..20]]

]

where ul = boxToWorld (0,19)

ur = boxToWorld (9,19)

bl = boxToWorld (0,0)

br = boxToWorld (9,0)

light\_gray :: Color

light\_gray = makeColor 0.8 0.8 0.8 1

gray :: Color

gray = makeColor 0.6 0.6 0.6 1

-- 描きたいものの場所を決める

boxToWorld :: (Int,Int) -> (Float,Float)

boxToWorld (i,j) =

(fromIntegral i\*boxWidthWorld-50,

fromIntegral j\*boxWidthWorld+boxWidth\_half)

-- ミノを表示するときの倍率

boxWidthWorld :: Float

boxWidthWorld = 5.0

-- ミノの辺の半分の長さの倍率。ミノを表示するうえで中心から上下左右にずらすときに使う

boxWidth\_half :: Float

boxWidth\_half = boxWidthWorld/2

-- ミノを表示するときの大きさに変換する

boxToWorldRel :: (Int,Int) -> (Float,Float)

boxToWorldRel (i,j) =

(fromIntegral i\*boxWidthWorld, fromIntegral j\*boxWidthWorld)

-- 初期のミノが何もない状態

emptyBox :: Box

emptyBox = M.fromList [((i,j),emptyColor) | i <- [0..9], j <- [0..20]]

-- ミノの回転に関する実装

adjustMinoPos :: Mino -> Mino

adjustMinoPos mino@Mino{..} =

let minX = minimum $ map fst mshape

minY = minimum $ map snd mshape

in mino { mshape = map (point\_add (-minX,-minY)) mshape }

-- 時計回り

spinCW :: (Int,Int) -> (Int,Int)

spinCW (x,y) = (y,-x)

rotateCWprim :: Mino -> Mino

rotateCWprim mino@Mino{..} =

mino { mshape = map spinCW mshape }

rotateCW :: Mino -> Mino

rotateCW = adjustMinoPos . rotateCWprim

-- 反時計回り

spinUCW :: (Int,Int) -> (Int,Int)

spinUCW (x,y) = (-y,x)

rotateUCWprim :: Mino -> Mino

rotateUCWprim mino@Mino{..} =

mino { mshape = map spinUCW mshape }

rotateUCW :: Mino -> Mino

rotateUCW = adjustMinoPos . rotateUCWprim

-- ライン消去の実装

-- 全部埋まっているラインを見つける

findFullLine :: Box -> Maybe Int

findFullLine box =

foldr (\x y -> Just x) Nothing

[j | j <- [0..19], all(\i -> box ! (i,j) /= emptyColor) [0..9]]

-- 埋まっている行を複数個一気に見つける

findFullLines :: Box -> [Int]

findFullLines box =

[j | j <- [0..19], all(\i -> box ! (i,j) /= emptyColor) [0..9]]

-- そろったラインを消す実装

collapseLines :: [Int] -> Box -> Box

collapseLines fullLines box =

foldr (\(i,j) b ->

if j `elem` fullLines then M.insert (i,j) emptyColor b -- ラインを消す

else M.insert (i,j) (box ! (i,j) ) b) -- 他はそのまま

M.empty $ [(i,j) | i <- [0..9], j <- [0..19]]

-- 下からせり上がるおじゃま

upLines :: Int -> Int -> Box -> Box

upLines lines empty box =

let box' = foldr (\(i,j) b ->

if j < lines && i == empty then M.insert (i,j) emptyColor b -- 下の行で1マス空けるところ

else if j < lines then M.insert (i,j) light\_gray b -- 下の行でおじゃまで埋めるところ

else M.insert (i,j) (box ! (i,j-lines)) b) -- n行下をコピー

M.empty $ [(i,j) | i <- [0..9], j <- [0..19]]

in box'

-- 上の行をコピーする実装

copyLines :: [Int] -> GameState -> [Int] -> Box

copyLines fullLines GameState{..} sp =

foldr (\(i,j) b ->

if j >= 20 - length fullLines then M.insert (i,j) emptyColor b -- 一番上の行は必ず消す

else if pair == 1 && length sp >= 1 && j == sp !! 0 then M.insert (i,j) (box ! (i,j+2)) b

else if pair == 2 && length sp >= 1 && j == sp !! 0 then M.insert (i,j) (box ! (i,j+1)) b

else if pair == 3 && length sp >= 1 && j == sp !! 0 then M.insert (i,j) (box ! (i,j+1)) b

else if pair == 4 && length sp >= 1 && j == sp !! 0 then M.insert (i,j) (box ! (i,j+1)) b

else if pair == 4 && length sp >= 2 && j == sp !! 1 then M.insert (i,j) (box ! (i,j+1)) b

else if j >= fullLines !! 0 then M.insert (i,j) (box ! (i,j+length fullLines)) b -- ラインができてる行よりも上は１つ上の行をコピー

else M.insert (i,j) (box ! (i,j) ) b) -- 下はそのまま

M.empty $ [(i,j) | i <- [0..9], j <- [0..19]]

type World = Freer ()

drawWorld :: World -> Picture

drawWorld (Effect \_ p \_ \_) = translate 10 (-240) . scale 4 4 $ p

drawWorld \_ = blank

-- キー入力でテトリミノを動かす

eventHandler :: Event -> World -> World

eventHandler (EventKey (Char c) Up \_ \_) (Effect \_ \_ KeyReq k) = k (Just [c])

-- 上下左右キーとEnterでもミノの移動が可能

eventHandler (EventKey (SpecialKey KeyUp) Up \_ \_) (Effect \_ \_ KeyReq k) = k (Just "up")

eventHandler (EventKey (SpecialKey KeyDown) Up \_ \_) (Effect \_ \_ KeyReq k) = k (Just "down")

eventHandler (EventKey (SpecialKey KeyRight) Up \_ \_) (Effect \_ \_ KeyReq k) = k (Just "right")

eventHandler (EventKey (SpecialKey KeyLeft) Up \_ \_) (Effect \_ \_ KeyReq k) = k (Just "left")

eventHandler (EventKey (SpecialKey KeyEnter) Up \_ \_) (Effect \_ \_ KeyReq k) = k (Just "drop")

eventHandler (EventKey (SpecialKey KeySpace) Up \_ \_) (Effect \_ \_ KeyReq k) = k (Just "spin")

eventHandler \_ w = w

-- 時間で操作中のテトリミノを下に落とす。今回はミノが下に着いたときのみ使用する

frameHandler :: Time -> World -> World

frameHandler \_ (Effect t p Pause k) | t > long\_timeout = k () -- pauseされたとき

frameHandler \_ (Effect t p KeyReq k) | t > short\_timeout = k Nothing -- getKeyされたとき

frameHandler t (Effect t' p r k) = Effect (t+t') p r k

frameHandler \_ w = w

-- pauseの時間

long\_timeout :: Float

long\_timeout = 0.3

-- getKeyの時間

short\_timeout :: Float

short\_timeout = 0.001 -- 秒

---------------------------------------------------------------------------

-- Hangmanで使ったIOモナドのようなもの

data Freer a where

Pure :: a -> Freer a

Effect :: Time -> Picture -> Req x -> (x -> Freer a) -> Freer a

instance Functor Freer where

fmap f (Pure x) = Pure $ f x

fmap f (Effect t p r k) = Effect t p r (fmap f . k)

instance Applicative Freer where

pure = Pure

Pure f <\*> m = fmap f m

Effect t p r k <\*> m = Effect t p r (\x -> k x <\*> m)

instance Monad Freer where

return = Pure

Pure x >>= k = k x

Effect t p r k' >>= k = Effect t p r (k' >>> k)

-- 副作用を持つ関数を合成する (CBV 合成)

(>>>) :: Monad m => (a -> m b) -> (b -> m c) -> (a -> m c)

f >>> g = (>>= g) . f

data Req a where

Pause :: Req ()

KeyReq :: Req (Maybe String)

-- Key入力を受け取って、出力する。getLineのようなもの

getKey :: GameState -> Picture -> Freer (Maybe String)

getKey GameState{..} extrap = do

let pict = pictures [drawBox box, extrap]

Effect 0 pict KeyReq pure

-- StateとPictureを出力する。PutStrLnのようなもの。画面が一瞬止まる

pause :: GameState -> Picture -> Freer ()

pause GameState{..} extrap = do

let pict = pictures [drawBox box, extrap]

Effect 0 pict Pause pure

-- ------------------------------------------------------------------------