511 lines (389 sloc) | 14.8 KB

```python
#追加したい物　自分の足跡をつける機能を追加する
#壁を0　#通路を1
import tkinter
import random
from tkinter import messagebox
import time

# キャンバスのサイズ設定
CANVAS_WIDTH = 1600 #元サイズ1600
CANVAS_HEIGHT = 900 #元サイズ900

# 迷路のサイズ設定#
WIDTH = 41
HEIGHT = 23

# 色設定
PATH_COLOR = "white"              #迷路の色
WALL_COLOR = "MidnightBlue"       #壁の色
GOAL_COLOR = "blue"               #ゴールの色
START_COLOR = "red"               #スタートの色
PASSED_COLOR = "orange"           #海藤表示時の経路の色
NOW_COLOR = "SkyBlue"             #現在地の色
AFTER_COLOR = "SpringGreen"       #移動後の色　#三島

# 数値の定義
PATH = 0
WALL = 1
GOAL = 2
START = 3
PASSED = 4
NOW = 5
AFTER = 6                         #三島

UP = 0
DOWN = 1
LEFT = 2
RIGHT = 3

```

```python
class Maze():
    def __init__(self, master):
        '''迷路ゲームの起動'''

        # ゲームを作成する親ウィジェット
        self.master = master

        # 迷路のサイズ
        self.width = WIDTH
        self.height = HEIGHT

        # 迷路の元になるリスト
        self.maze = None

        # 現在位置
        self.now = None

        #過ぎた後の位置            #三島
        self.after = None

        # 1つ前の位置
        self.before = None


        # スタートとゴールの位置
        self.start = None
        self.goal = None

        # 解答を既に見つけたかどうかのフラグ
        self.resolved = False

        # 迷路の元になる２次元リストを作成
        self.createMaze()

        # ウィジェットを作成して迷路を表示
        self.createWidgets()

    def createMaze(self):
        '''迷路の元になる２次元リストを作成'''

        # ２次元リストを作成（全て壁）
        self.maze = [[WALL] * self.width for j in range(self.height)]

        # 開始点を決定
        i = 2 * random.randint(0, self.width // 2 - 1) + 1
        j = 2 * random.randint(0, self.height // 2 - 1) + 1

        # (i, j) を通路に設定
        self.maze[j][i] = PATH

        # 穴掘り法でマス(i, j) を起点に穴を掘る
        self.dig(i, j)

        # ここまで穴掘り法

        # スタートを設定
        self.setStart()

```

```
97          # ゴールを決定
98          self.setGoal()
99
100     def setStart(self):
101         '''スタートの位置を設定'''
102
103         # 通路の数をカウント
104         num_path = 0
105         for j in range(self.height):
106             for i in range(self.width):
107                 if self.maze[j][i] == PATH:
108                     num_path += 1
109
110         # スタートの位置をランダムに決定
111         startPos = random.randint(0, num_path - 1)
112
113         # 左上からstartPos個目の通路のマスをスタートに設定
114         count = 0
115         for j in range(self.height):
116             for i in range(self.width):
117                 if self.maze[j][i] == PATH:
118                     if count == startPos:
119                         self.maze[j][i] = START
120                         self.start = (i, j)
121                         return
122                     else:
123                         count += 1
124
125     def setGoal(self):
126         '''ゴールの位置を設定'''
127
128         # 通路の数をカウント
129         num_path = 0
130         for j in range(self.height):
131             for i in range(self.width):
132                 if self.maze[j][i] == PATH:
133                     num_path += 1
134
135         # ゴールの位置をランダムに決定
136         goalPos = random.randint(0, num_path - 1)
137
138         # 左上からgoalPos個目の通路のマスをゴールに設定
139         count = 0
140         for j in range(self.height):
141             for i in range(self.width):
142                 if self.maze[j][i] == PATH:
143                     if count == goalPos:
144                         self.maze[j][i] = GOAL
145                         self.goal = (i, j)
146                         return
147                     else:
148                         count += 1
149
150     def dig(self, i, j):
151         '''(i,j)座標を起点に穴を掘る'''
152
153         # どの方向を掘ろうとしたかを覚えておく変数
154         up = True
```

```
        down = True
        left = True
        right = True

        # 全方向試すまでループ
        while up or down or left or right:
            # 0 - 3 の乱数を取得
            d = random.randint(0, 3)

            if d == UP:
                # 上方向が掘れるなら掘る
                if j - 2 >= 0 and j - 2 < self.height:
                    if self.maze[j - 2][i] == WALL:
                        self.maze[j - 2][i] = PATH
                        self.maze[j - 1][i] = PATH
                        self.dig(i, j - 2)

                    up = False

            elif d == DOWN:
                # 下方向が掘れるなら掘る
                if j + 2 >= 0 and j + 2 < self.height:
                    if self.maze[j + 2][i] == WALL:
                        self.maze[j + 2][i] = PATH
                        self.maze[j + 1][i] = PATH
                        self.dig(i, j + 2)

                    down = False

            elif d == LEFT:
                # 左方向が掘れるなら掘る
                if i - 2 >= 0 and i - 2 < self.width:
                    if self.maze[j][i - 2] == WALL:
                        self.maze[j][i - 2] = PATH
                        self.maze[j][i - 1] = PATH
                        self.dig(i - 2, j)

                    left = False

            elif d == RIGHT:
                # 右方向が掘れるなら掘る
                if i + 2 >= 0 and i + 2 < self.width:
                    if self.maze[j][i + 2] == WALL:
                        self.maze[j][i + 2] = PATH
                        self.maze[j][i + 1] = PATH
                        self.dig(i + 2, j)

                    right = False

    def change_color(self, i, j):
        '''(i,j)座標に対応する長方形の色を変更'''

        # mazeリストの値に応じて色を取得
        if self.maze[j][i] == WALL:
            color = WALL_COLOR
        elif self.maze[j][i] == PATH:
            color = PATH_COLOR
        elif self.maze[j][i] == GOAL:
```

```
            color = GOAL_COLOR
        elif self.maze[j][i] == START:
            color = START_COLOR
        elif self.maze[j][i] == PASSED:
            color = PASSED_COLOR
        elif self.maze[j][i] == NOW:
            color = NOW_COLOR
        elif self.maze[j][i] == AFTER:  #三島
            color = AFTER_COLOR
        else:
            print("そんなマスはあり得ません")
            return

        # (i,j)座標の長方形を特定するためにタグを作る
        tag = "rectangle_" + str(i) + "_" + str(j)

        # そのタグがつけられたfill設定を変更
        self.canvas.itemconfig(
            tag,
            fill=color
        )

    def createWidgets(self):
        '''ウィジェットを作成する'''

        # キャンバスウィジェットの作成と配置
        self.canvas = tkinter.Canvas(
            self.master,
            width=CANVAS_WIDTH,
            height=CANVAS_HEIGHT,
        )
        self.canvas.pack()

        for j in range(self.height):
            for i in range(self.width):

                # 後から操作できるように座標に基づいたタグを付ける
                tag = "rectangle_" + str(i) + "_" + str(j)

                # キャンバスへの長方形の描画（迷路の描画）
                self.canvas.create_rectangle(
                    3 + i * CANVAS_WIDTH / self.width,
                    3 + j * CANVAS_HEIGHT / self.height,
                    3 + (i + 1) * CANVAS_WIDTH / self.width,
                    3 + (j + 1) * CANVAS_HEIGHT / self.height,
                    width=0,  # 枠線なし
                    tag=tag  # タグ
                )

                # 長方形に色をつける
                self.change_color(i, j)

        btn = tkinter.Button(app, text="answer", command=self.show_answer, height=1,width=5)
        btn.place(x=700, y=5)  #丸山

    def resolve_maze(self, i, j):
        '''(i,j)マスから移動できる方向に１マス進む'''

```

```
271         # 迷路外 or 壁のマスが指定された場合はエラー
272         if i < 0 or i >= self.width or j < 0 or j >= self.height or self.maze[j][i] == WALL:
273             return
274 
275         # 既に経路表示済みの場合は即終了
276         if self.resolved:
277             return
278 
279         # このマスがゴールなら終了
280         if self.maze[j][i] == GOAL:
281 
282             # ここまでの経路を表示
283             self.print_pass()
284             self.resolved = True
285             return
286 
287         # このマスを通過したことを覚えておく
288         if self.maze[j][i] != START:
289             self.maze[j][i] = PASSED
290 
291         # 上に１マス移動
292         ni = i
293         nj = j - 1  # 上に移動
294         if nj >= 0:
295             if self.maze[nj][ni] != WALL:
296                 if self.maze[nj][ni] != PASSED and self.maze[nj][ni] != START:
297                     # 次のマスからゴールまで移動させる
298                     self.resolve_maze(ni, nj)
299 
300         # 下に１マス移動
301         ni = i
302         nj = j + 1  # 下に移動
303         if nj < self.height:
304             if self.maze[nj][ni] != WALL:
305                 if self.maze[nj][ni] != PASSED and self.maze[nj][ni] != START:
306                     # 次のマスからゴールまで移動させる
307                     self.resolve_maze(ni, nj)
308 
309         # 左に１マス移動
310         ni = i - 1  # 左に移動
311         nj = j
312         if ni >= 0:
313             if self.maze[nj][ni] != WALL:
314                 if self.maze[nj][ni] != PASSED and self.maze[nj][ni] != START:
315                     # 次のマスからゴールまで移動させる
316                     self.resolve_maze(ni, nj)
317 
318         # 右に１マス移動
319         ni = i + 1  # 右に移動
320         nj = j
321         if ni < self.width:
322             if self.maze[nj][ni] != WALL:
323                 if self.maze[nj][ni] != PASSED and self.maze[nj][ni] != START:
324                     # 次のマスからゴールまで移動させる
325                     self.resolve_maze(ni, nj)
326 
327         # このマスを通過したことを忘れる
328         if self.maze[j][i] != START:
```

```
            self.maze[j][i] = PATH

    def print_pass(self):
        '''答えを表示する'''

        for j in range(self.height):
            for i in range(self.width):
                self.change_color(i, j)


    def show_answer(self):
        '''解答表示する'''

        if self.playing:

            # プレイ中フラグをFalseに設定
            self.playing=False

            # 答えを見つけ出して表示する
            self.resolve_maze(self.start[0], self.start[1])

    def play(self):
        '''ゲームプレイを開始する'''

        # ゲームプレイフラグをTrueにセット
        self.playing = True

        # 現在地をスタート値値に設定
        self.now = self.start

        # 上下左右キーに対してイベント受付設定
        self.master.bind("<KeyPress-Up>", self.up_move)
        self.master.bind("<KeyPress-Down>", self.down_move)
        self.master.bind("<KeyPress-Left>", self.left_move)
        self.master.bind("<KeyPress-Right>", self.right_move)

#ここに移動後の色を追加する
    def update(self):
        '''移動後の状態に迷路リストを更新'''

        # 移動後の現在地を取得
        i, j = self.now
        #i, j = self.after

        # GOALであれば終了処理
        if self.maze[j][i] == GOAL:
            self.game_clear()
            return

        # 現在地を更新
        self.maze[j][i] = NOW

        # 色を更新
        self.change_color(i, j)

       ##### # 移動前の現在地を取得
        i, j = self.before

```

```
387         # 移動前の位置を更新
388         if self.before != self.start:
389             self.maze[j][i] = AFTER     #三島
390         else:
391             self.maze[j][i] = START
392 
393         # 色を更新
394         self.change_color(i, j)
395 
396     def up_move(self, event):
397         ''' 上に１マス移動する'''
398 
399         # 現在地を取得
400         now = self.now
401         i, j = now
402 
403         # 上に移動
404         j = j - 1
405 
406         # 迷路外 or 壁のマスが指定された場合は移動しない
407         if i < 0 or i >= self.width or j < 0 or j >= self.height or self.maze[j][i] == WALL:
408             return
409 
410         self.before = self.now
411 
412         # 移動後の座標を現在位置に設定
413         self.now = i, j
414 
415         self.update()
416 
417     def down_move(self, event):
418         ''' 下に１マス移動する'''
419 
420         # 現在地を取得
421         now=self.now
422         i, j=now
423 
424         # 下に移動
425         j=j + 1
426 
427         # 迷路外 or 壁のマスが指定された場合は移動しない
428         if i < 0 or i >= self.width or j < 0 or j >= self.height or self.maze[j][i] == WALL:
429             return
430 
431         self.before=self.now
432 
433         # 移動後の座標を現在位置に設定
434         self.now=i, j
435 
436         self.update()
437 
438     def left_move(self, event):
439         ''' 左に１マス移動する'''
440 
441         # 現在地を取得
442         now=self.now
443         i, j=now
444 
```

```
445         # 左に移動
446         i=i - 1
447 
448         # 迷路外 or 壁のマスが指定された場合は移動しない
449         if i < 0 or i >= self.width or j < 0 or j >= self.height or self.maze[j][i] == WALL:
450             return
451 
452         self.before=self.now
453 
454         # 移動後の座標を現在位置に設定
455         self.now=i, j
456 
457         self.update()
458 
459     def right_move(self, event):
460         ''' 右に１マス移動する'''
461 
462         # 現在地を取得
463         now=self.now
464         i, j=now
465 
466         # 右に移動
467         i=i + 1
468 
469         # 迷路外 or 壁のマスが指定された場合は移動しない
470         if i < 0 or i >= self.width or j < 0 or j >= self.height or self.maze[j][i] == WALL:
471             return
472 
473         self.before=self.now
474 
475 
476         # 移動後の座標を現在位置に設定
477         self.now=i, j
478 
479         # 座標に移動する
480         self.update()
481 
482 
483     def game_clear(self):
484         self.playing=False
485 
486         self.canvas.create_text(
487             CANVAS_WIDTH // 2,
488             CANVAS_HEIGHT // 2,
489             fill="red",
490             font=("@Terminal", 100),
491             text="GAME CLEAR!!",
492 
493         )
494         # ゲーム開始からゴール到着までの時間を測定、 （遠藤）
495         end = time.time() - begin
496         # ゴール後に経過時間を示したメッセージボックスが出る。　　（遠藤）
497         messagebox.showinfo("GOAL", f"{end}秒かかりました。")
498 
499         self.master.unbind("<KeyPress-Up>")
500         self.master.unbind("<KeyPress-Left>")
501         self.master.unbind("<KeyPress-Right>")
502         self.master.unbind("<KeyPress-Down>")
```

```
503
504 app=tkinter.Tk()
505 app.title("フレゼミの女！！！！！")    #aoi
506
507 maze=Maze(app)
508 begin = time.time()
509 maze.play()
510
511 app.mainloop()
```