Contents

1 introduction **1**

2 Basic idea **1**

3 "Races" **1**

4 Classes **1**

5 Stats **2**
5.1 List of stats to consider: 2

6 Code **2**

7 Globals **3**

8 Cell function **3**

9 Ressource handler **3**

10 Sprite class **4**

11 Window handler **4**

12 View MIGHT BE REMOVED **4**

13 Drawing **5**

14 Map **6**

15 Game Loop function **7**

16 Player **8**

17 Input player movement **8**

18 Move **9**

19 Main **9**

1 introduction

2 Basic idea

A dungeon crawler set in a modern day office space (with cubicles, cantinas etc.) instead of fighting I need another way of handling engagement.

3 "Races"

- Zzzombie - the constantly sleepy coworker, never seems to be quite awake nor have they had a good nights sleep in their life.
- Gobbo - The coworker who handles mails, you're not sure if they've ever had a shower, but the greasy strands of hair makes you think not.
- Creeps -
- Emotional vampire
- Ghost
- Troglodytes

4 Classes

- Financial
- IT
- Assistant
- Intern
- Secretary
- Janitor
- Boss

5 Stats

Stats in an RPG is like... something... really... hmm (ed: come back with a zinger here). Usually the stats in a traditional roguelike might be something akin to - Constitution, Strength, Wisdom, Intelligence etc. basically if you've seen a rulebook for a TTRPG you can probably recognize most of these AND THEY WORK! that's why people use them, they really REALLY work. I'm not choosing differently because I think I can do better, but because, well since the game is already set in a modern office environment, there's little reason to not have some weird quirk with the stats again.

5.1 List of stats to consider:

- Neuroticism: how panicked and alert a player is about their surrounding.
- Openness: how open they can allow themself to be.
- Reasoning: how good they are at logically deducting things, (maybe reasoning and neuroticism can kindda benefit and be a detriment at the same time)
- Madness: How mad the character is (remember! you don't have to be mad to work here, but it helps!)
- Agreeableness: How much other characters like you.
- Laziness: How much (or little :/) energy a character has.

I Really REALLY wanted to put "Agility" into this stat list, but not agility as it is traditionally used, but more as in the agile method that is usually claimed by every company under the sun that has too much money and not enough sensible managers/CEOs. Though I feel like that'll be an unnescessary dig at alot of people I have tremendous respect for. (though maybe they don't care what a nobody on the internet does or say... I dunno :))

So that's the N.O.R.M.A.L. stat system (you'd think I'd have picked these words carefully... and you'd be right) patent pending yadda yadda.

6 Code

How neat would it be if we could quit here? alas now we have to try and code this :S

```
from __future__ import annotations
from abc import ABC, abstractmethod
import pygame
import pprint
from pygame.locals import *
import numpy
from collections.abc import Callable
```

7 Globals

Global variables are frowned upon by virtually everyone and their mom, which is fair, I don't like them either, but I don't have a better idea.

Global variables are the following

- cell size, i.e. how large the cell that fills the screen surface should be. Then I can later just divide the screenwidth or screenheight of the drawing destination surface by the cell size to know how big the are is. (so I don't draw outside the screen/surface), this is also where I'm placing the filenames for fonts which I might need/want to change.

```
CELLSIZE = 32
FONTFILE = "terminal8x8_gs_ro.png"
```

8 Cell function

The cell function handles converting the direction vectors into coordinates that can

9 Ressource handler

```
class FileLoader():
    _fontimage : pygame.Surface
    def __init__(self, filename : str):
        self._fontimage = pygame.image.load("terminal8x8_gs_ro.png")
        self._fontimage.set_colorkey((0,0,0))

    def GetTileSet(self):
        pass
```

10 Sprite class

```
class TileSheet:
    _tiles = [pygame.Surface]
    def __init__(self,
                 file : pygame.Surface,
                 width : int,
                 height : int,
                 rows : int,
                 columns : int):
        for x in range(rows):
            for y in range(columns):
                self._tiles.append(
                    pygame.transform.scale(
                        file.subsurface(y * width, x * height, width, height),
                        (CELLSIZE, CELLSIZE)
                    )
                )
```

11 Window handler

The window class is where the initializing is going to happen, as well as where the pygame window.

```
class Window():
    _size = (int, int)
    _surface = pygame.Surface
    def __init__(self, width, height) -> None:
        self._surface = pygame.display.set_mode((width, height), 0, 32)


    def surface(self) -> pygame.Surface:
        return self._surface
```

12 View MIGHT BE REMOVED

The view holds a surface and a pygame.Rect. The rect is moved around to "slice" a subsurface from the map.

```
class View:
```

```
    _camera : pygame.Rect
    _trackingObject = None
    def __init__(self,
                 topx : int,
                 topy : int,
                 cameraWidth : int,
                 cameraHeight : int,
                 trackingObject = None):
        self._camera = pygame.Rect(topx, topy, topx+cameraWidth, topycameraHeight)
        if trackingObject is not None:
            self._trackingObject = trackingObject

    def slice(self, surface : pygame.Surface):
        return surface.Subsurface(self._camera)

    def trackObject(self, surface : pygame.Surface):
        pass

    def checkCenterObject(self):
        pass

    def move(self):
        if self._trackingObject is not None:
            while(self._camera.centerx > self._trackingObject.x):
                self._camera.centerx -= 1
            while(self._camera.centerx < self._trackingObject.x):
                self._camera.centerx += 1
            while(self._camera.centery > self._trackingObject.y):
                self._camera.centery -= 1
            while(self._camera.centery < self._trackingObject.y):
                self._camera.centery += 1

    def update(self) -> pygame.Surface:
        pass
```

13 Drawing

Drawmap function is only called to draw the surface of the static map.

```
def drawMap(map : str,
```

```
            pos : [(int, int)],
            tiles : [pygame.Surface],
            destination :pygame.Surface):
    _drawingList : [(pygame.Surface,(int,int))] = []
    x = 0
    y = 0
    for ind, c in enumerate(map):
        _drawingList.append((tiles[ord(c)+1], pos[ind]))
    destination.blits(_drawingList)
```

Drawing

```
def drawing(chars : str,
            pos : [(int, int)],
            tiles : [pygame.Surface],
            destination :pygame.Surface):
    _drawingList : [(pygame.Surface,(int,int))] = []
    x = 0
    y = 0
    for ind, c in enumerate(chars):
        _drawingList.append((tiles[ord(c)+1], pos[ind]))
    destination.blits(_drawingList)
```

14 Map

the map is for now just a container class for a premade "dungeon", this is to test whether or not the drawing function can handle the sheer drawing calls. AND that it can handle the various characters.

It's supposed to just have a giant, static (more or less static) image of the map.

```
class Map:
    _str : str
    _pos : [(int,int)] = []
    _map : pygame.Surface
    def __init__(self, tiles):
        # TEMP map
        self._str = ""
        tempMap = [ "##################################",
                    "#       #             #        #",
```

```
                "#       #             #        #",
                "#       #             #        #",
                "#       #             #        #",
                "#                               #",
                "#                               #",
                "#           y                   #",
                "#      Hello                    #",
                "#       p                       #",
                "#                               #",
                "#                               #",
                "#                               #",
                "#                               #",
                "#                               #",
                "#################################"
        ]
        w = len(tempMap[0]) * CELLSIZE
        h = len(tempMap) * CELLSIZE
        self._map = pygame.Surface((w, h))

        for i, s in enumerate(tempMap):
            self._str = self._str + s
            for string_i, _ in enumerate(s):
                self._pos.append((string_i*CELLSIZE, i*CELLSIZE))
        drawMap(self._str, self._pos, tiles, self._map)

    def map(self):
        return self._map
```

15 Game Loop function

The Game loop is where the structure of the game is at.

```
def GameLoop(window, map, player, tiles):
    out = True
    enteties = []
    while(out):
        movVec = (0,0)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                out = False
```

```
            elif event.type == pygame.KEYDOWN:
                movVec = getInput(event)
        if player.arrived():
            player._next_x += movVec[0] * CELLSIZE
            player._next_y += movVec[1] * CELLSIZE
        move(player)
        window.surface().blit(map.map(), (0,0))
        drawing(player._c, [(player._x, player._y)], tiles._tiles, window.surface())
        pygame.display.flip()
    pprint.pprint(str(player._x) + ", " + str(player._y) + ", " + str(player._next_x) +
    #pprint.pprint(map._pos)
        # Run game
```

16 Player

The player class is, for now just a place holder, keeping the player char (literally a char value), position and that's it.

```
class Player:
    _next_x : int
    _x : int
    _next_y : int
    _y : int
    _c : chr
    def __init__(self, x : int, y : int, char : chr):
        self._x = x
        self._y = y
        self._next_x = x
        self._next_y = y
        self._c = char
    def arrived(self) -> bool:
        if self._x == self._next_x or self._y == self._next_y:
            return True
        return False
```

17 Input player movement

for now I just handle the input through a simple function that checks whether or not a valid key has been pressed.

```
def getInput(ev):
    inputList = { pygame.K_UP : (0,-1),
                  pygame.K_DOWN : (0,1),
                  pygame.K_LEFT : (-1, 0),
                  pygame.K_RIGHT : (1, 0),
    }
    return inputList.get(ev.key, (0,0))
```

18 Move

Move function is just meant to update a players position to the new position.

```
def move(player):
    if player._next_x > player._x:
        player._x += 1
    elif player._next_x < player._x:
        player._x -= 1
    elif player._next_y > player._y:
        player._y += 1
    elif player._next_y < player._y:
        player._y -= 1
```

19 Main

In the main function I initialize the different components, like the window, the gameloop function, etc.

```
def main():
    pygame.init()
    pygame.font.init()
    # -------
    _files = FileLoader("terminal8x8_gs_ro.png")
    _tiles = TileSheet(_files._fontimage, 8, 8, 16, 16)
    window = Window(800, 600)
    map = Map(_tiles._tiles)
    player = Player(32, 32, "@")
    GameLoop(window, map, player, _tiles)
    # -------
    pygame.quit()
```

```
if __name__=="__main__":
    main()
```