Odd Semester (2020)



**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Assignment Cover Letter**

**(Individual Work****)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | |  | |
| **Student Information**: **Surname** | | | | | **Given Names**  **Jordy** | | **Student ID Number**  **2101693542** | |
| 1. | | **Filbert** |  | |
|  |  |
| **Course Code** | **: COMP6502** |  |  | | **Course Name** | | **: Introduction to Programming** | |
| **Class** | **: L1BC** |  |  | | **Name of Lecturer(s)** | | **:** 1. Jude Martinez | |
|  |  |  |  | |  | | 2. Minaldi Loeis | |
| **Major** | **: CS** |  |  | |  | |  | |
| **Title of Assignment**  (if any) | : SandboxSandbox | |  |  | |  | |  | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  |  | |  | |  | |
| **Due Date** | **: 6-10-2017** |  |  | | **Submission Date** | | **: 6-10-2017** | |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

# Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1. Veronika Stephanie

**“RandomRandom”**

**Name :Jordy Filbert**

**ID :2101693542**

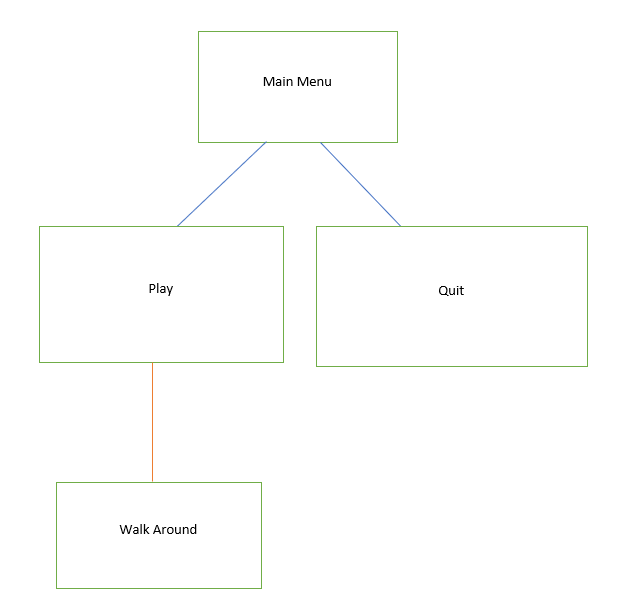
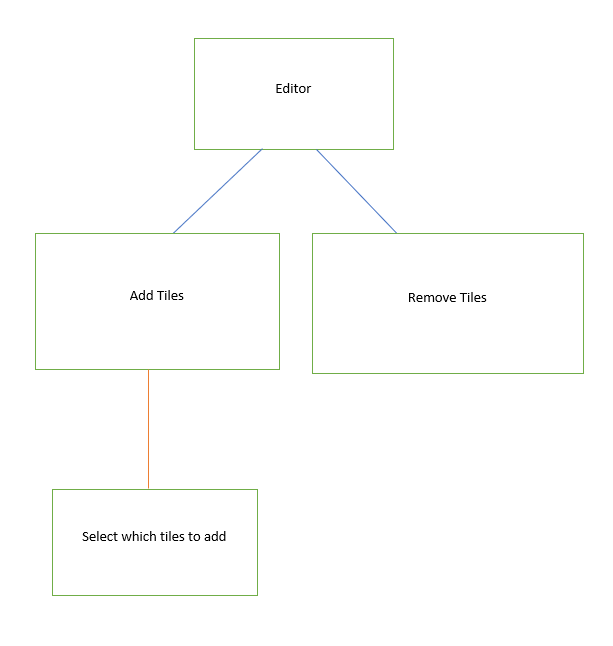
1. **Description**

**The function of this program:**

This game(Program) is made so people can create tiles and edit tiles. The game has no perfect purpose, it is just a game made so people can waste time doing something boring such as this stupid game.

**II.a. Design/Plan**

**Project’s Hierarchy Chart**

 **Main Game Map Editor**

**II.b. Explanation of Each function**

**The Main Game: ( *main.py* )**

1. **Outside the main loop of the game:**

* Bunch of stuff is imported here
  + pygame, sys, time, math
  + and the whole scripts folder
* Many variables are set to 0 as it’s initial value
* Fps counter created
* Some define functions are created to do the main stuff in the game(such as exit and play button)
* Map and logo is loaded from here
* Player starting position is settled here
* isRunning is set to True (isRunning is just a variable I made to know if the game is running or not)

1. **Inside main loop of the game**

* Key mapping is created here(up down left right etc)
* The game renders all the texture here
* Menu selection is also done here

**Map Editor : (map\_editor.py)**

* Bunch of scripts are imported here.
  + Pygame, sys, math
  + scripts.UltraColor, scripts.textures

**1.Outside the main loop of the map\_editor**

* defines are created to make the process easier later.
  + Export\_map
  + Load\_map
* The window is created and implemented here
* Brush is set to 1 as default(grass)
* Selector properties are created here

**2.inside the main loop of the map\_editor**

* Button mapping are created here
* The terrain are loaded here
* Fps is maxed to 60

**Inside the scripts folder:**

* Globals
* Map\_engine
* Meloonatic\_gui
* NPC
* Player
* Textures
* UltraColor

**a.globals.py**

* There is pretty much nothing here, just the main variable that applies to *main.py* and *map\_editor.py*

**b.map\_engine.py**

* Map engine is used to load map for the main game
* It shows the title of the map

**c.meloonatic.py**

* This is a GUI I downloaded of the internet (credit is inside the file)

**d.NPC.py**

* This file is to define where the character face is showing
  + North, east, west, south

**e.player.py**

* This loads the character (.png)
* It renders where it stands and where it face

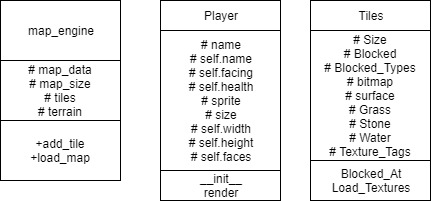
**f.textures.py**

* The texture file defines what is blocked and what is not
* It loads the textures of the game
  + Grass, stone, water,etc
* It defines the tag as number

**g.UltraColor.py**

* This is a file I downloaded of Meloonatic, this file only make life easier so we don’t have to insert all the rgb color all the time, just the name of the color.

**UML Diagram:**

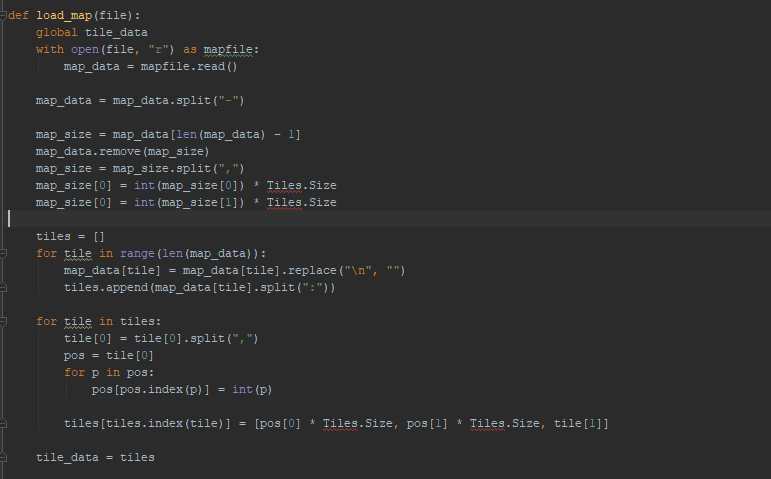
****

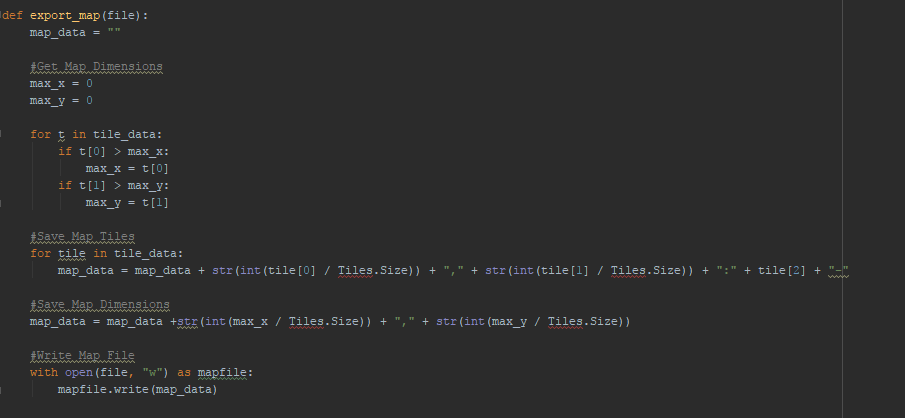
**III.a. Lessons that Have Been Learned**

1st of November,

The first time I tried to make this project, it took it very difficult for me especially the part where I mark what is the coordinate of the box, and how to set where the character face. I asked couple of guys on the internet on how it work, and they taught me how to do it.

The most frustrating part of the program is how you load the map and export the map.





It is very difficult because you have to split spaces “,” and “-“ as their default position consist of those things.

**V. Source Code**

1. ***main.py***

***import pygame, sys, time, math***

***from scripts.UltraColor import \****

***from scripts.textures import \****

***from scripts.globals import \****

***from scripts.map\_engine import \****

***from scripts.NPC import \****

***from scripts.player import \****

***from scripts.meloonatic\_gui import \****

***pygame.init()***

***cSec = 0***

***cFrame = 0***

***FPS = 0***

***deltatime = 0***

***terrain = Map\_Engine.load\_map("maps\\world.map")***

***clock=pygame.time.Clock()***

***fps\_font = pygame.font.Font("C:\\Windows\Fonts\\Verdana.ttf", 20)***

***sky = pygame.image.load("graphics\\sky.png")***

***Sky = pygame.Surface(sky.get\_size(), pygame.HWSURFACE)***

***Sky.blit(sky,(0, 0))***

***del sky***

***logo\_img\_temp = pygame.image.load("graphics\\logo.png")***

***# logo\_img\_temp = pygame.transform.scale(logo\_img\_temp, (239, 176))***

***logo\_img = pygame.Surface(logo\_img\_temp.get\_size(), pygame.HWSURFACE)***

***logo\_img.blit(logo\_img\_temp, (0, 0))***

***del logo\_img\_temp***

***def show\_fps():***

***fps\_overlay = fps\_font.render(str(FPS), True, Color.Goldenrod)***

***window.blit(fps\_overlay, (0, 0))***

***def create\_window():***

***global window, window\_height, window\_width, window\_title***

***window\_width, window\_height = 800, 600***

***window\_title = 'Final Project'***

***pygame.display.set\_caption(window\_title)***

***window = pygame.display.set\_mode((window\_width, window\_height), pygame.HWSURFACE|pygame.DOUBLEBUF)***

***def count\_fps():***

***global cSec, cFrame, FPS, deltatime***

***if cSec == time.strftime("%S"):***

***cFrame += 1***

***else:***

***FPS = cFrame***

***cFrame = 0***

***cSec = time.strftime('%S')***

***if FPS > 0:***

***deltatime = 1 / FPS***

***create\_window()***

***player = Player("Grimon")***

***player\_w, player\_h = player.width, player.height***

***player\_x = ((window\_width / 2 - player\_w/ 2 - Globals.camera\_x) / Tiles.Size)***

***player\_y = ((window\_height / 2 - player\_h / 2 - Globals.camera\_y) / Tiles.Size)***

***#Initialize GUI***

***def Play():***

***Globals.scene = "game"***

***def Exit():***

***global isRunning***

***isRunning = False***

***btnPlay = Menu.Button(text = "Play", rect = (0, 0, 300, 60),***

***tag = ("menu", None))***

***btnPlay.Left = window\_width / 2 - btnPlay.Width / 2***

***btnPlay.Top = window\_height / 2 - btnPlay.Height / 2***

***btnPlay.Command = Play***

***btnExit = Menu.Button(text = "Exit", rect = (0, 0, 300, 60),***

***tag = ("menu", None))***

***btnExit.Left = btnPlay.Left***

***btnExit.Top = btnPlay.Top + btnExit.Height + 3***

***btnExit.Command = Exit***

***menuTitle = Menu.Text(text = "Welcome to the Game", color = Color.Cyan,***

***font = Font.Large)***

***menuTitle.Left, menuTitle.Top = window\_width / 2 - menuTitle.Width / 2, 0***

***logo = Menu.Image(bitmap = logo\_img)***

***logo.Left = window\_width / 2 - logo.Width / 2***

***logo.Top = window\_height / 2 - logo.Height / 2***

***isRunning = True***

***while isRunning:***

***for event in pygame.event.get():***

***if event.type == pygame.QUIT:***

***isRunning=False***

***if event.type == pygame.KEYDOWN:***

***if event.key == pygame.K\_w:***

***Globals.camera\_move=1***

***player.facing = "north"***

***if event.key == pygame.K\_s:***

***Globals.camera\_move=2***

***player.facing = "south"***

***if event.key == pygame.K\_a:***

***Globals.camera\_move=3***

***player.facing = "east"***

***if event.key == pygame.K\_d:***

***Globals.camera\_move=4***

***player.facing = "west"***

***elif event.type == pygame.KEYUP:***

***Globals.camera\_move = 0***

***elif event.type == pygame.MOUSEBUTTONDOWN:***

***if event.button == 1: #Left Click***

***#Handle button click events***

***for btn in Menu.Button.All:***

***if btn.Tag[0] == Globals.scene and btn.Rolling:***

***if btn.Command != None:***

***btn.Command() #DO BUTTON EVENT***

***btn.Rolling = False***

***break #EXIT LOOP***

***#Render Scene***

***if Globals.scene == "game":***

***#LOGIC***

***if Globals.camera\_move == 1:***

***if not Tiles.Blocked\_At((round(player\_x), math.floor(player\_y))):***

***Globals.camera\_y += 100 \* deltatime***

***elif Globals.camera\_move == 2:***

***if not Tiles.Blocked\_At((round(player\_x), math.ceil(player\_y))):***

***Globals.camera\_y -= 100 \* deltatime***

***elif Globals.camera\_move ==3:***

***if not Tiles.Blocked\_At((math.floor(player\_x), round(player\_y))):***

***Globals.camera\_x += 100 \* deltatime***

***elif Globals.camera\_move ==4:***

***if not Tiles.Blocked\_At((math.ceil(player\_x), round(player\_y))):***

***Globals.camera\_x -= 100 \* deltatime***

***player\_x = ((window\_width / 2 - player\_w / 2 - Globals.camera\_x) / Tiles.Size)***

***player\_y = ((window\_height / 2 - player\_h / 2 - Globals.camera\_y) / Tiles.Size)***

***#Render Graphics***

***window.blit(Sky, (0, 0))***

***window.blit(terrain, (Globals.camera\_x, Globals.camera\_y))***

***player.render(window, (window\_width / 2 - player\_w / 2,***

***window\_height /2 - player\_h / 2))***

***#PROCESS MENU***

***elif Globals.scene == "menu":***

***window.fill(Color.Fog)***

***logo.Render(window)***

***menuTitle.Render(window)***

***for btn in Menu.Button.All:***

***if btn.Tag[0] == "menu":***

***btn.Render(window)***

***show\_fps()***

***pygame.display.update()***

***count\_fps()***

***# clock.tick(30)***

***pygame.quit()***

***sys.exit()***

***map\_editor.py***

import pygame, sys, math

from scripts.UltraColor import \*

from scripts.textures import \*

def export\_map(file):

map\_data = ""

#Get Map Dimensions

max\_x = 0

max\_y = 0

for t in tile\_data:

if t[0] > max\_x:

max\_x = t[0]

if t[1] > max\_y:

max\_y = t[1]

#Save Map Tiles

for tile in tile\_data:

map\_data = map\_data + str(int(tile[0] / Tiles.Size)) + "," + str(int(tile[1] / Tiles.Size)) + ":" + tile[2] + "-"

#Save Map Dimensions

map\_data = map\_data +str(int(max\_x / Tiles.Size)) + "," + str(int(max\_y / Tiles.Size))

#Write Map File

with open(file, "w") as mapfile:

mapfile.write(map\_data)

def load\_map(file):

global tile\_data

with open(file, "r") as mapfile:

map\_data = mapfile.read()

map\_data = map\_data.split("-")

map\_size = map\_data[len(map\_data) - 1]

map\_data.remove(map\_size)

map\_size = map\_size.split(",")

map\_size[0] = int(map\_size[0]) \* Tiles.Size

map\_size[0] = int(map\_size[1]) \* Tiles.Size

tiles = []

for tile in range(len(map\_data)):

map\_data[tile] = map\_data[tile].replace("\n", "")

tiles.append(map\_data[tile].split(":"))

for tile in tiles:

tile[0] = tile[0].split(",")

pos = tile[0]

for p in pos:

pos[pos.index(p)] = int(p)

tiles[tiles.index(tile)] = [pos[0] \* Tiles.Size, pos[1] \* Tiles.Size, tile[1]]

tile\_data = tiles

window = pygame.display.set\_mode((1280, 720), pygame.HWSURFACE)

pygame.display.set\_caption("Map Editor")

clock = pygame.time.Clock()

txt\_font = pygame.font.Font("C:\\Windows\\Fonts\\Verdana.ttf", 20)

mouse\_pos = 0

mouse\_x, mouse\_y = 0, 0

map\_width, map\_height = 100 \* Tiles.Size, 100\* Tiles.Size

selector = pygame.Surface((Tiles.Size, Tiles.Size), pygame.HWSURFACE|pygame.SRCALPHA)

selector.fill(Color.WithAlpha(100, Color.CornflowerBlue))

tile\_data = []

camera\_x, camera\_y = 0, 0

camera\_move = 0

brush = "1"

#Initialize Default Map

for x in range(0, map\_width, Tiles.Size):

for y in range(0, map\_height, Tiles.Size):

tile\_data.append([x, y, "1"])

isRunning = True

while isRunning:

for event in pygame.event.get():

if event.type == pygame.QUIT:

isRunning = False

if event.type == pygame.KEYDOWN:

#Movement

if event.key == pygame.K\_w:

camera\_move = 1

elif event.key == pygame.K\_s:

camera\_move = 2

elif event.key == pygame.K\_a:

camera\_move = 3

elif event.key == pygame.K\_d:

camera\_move = 4

#BRUSHED

if event.key == pygame.K\_F4:

brush = "r"

elif event.key == pygame.K\_F1:

selection = input("Brush Tag: ")

brush = selection

#Save Map

if event.key == pygame.K\_F11:

name = input("Map Name: ")

export\_map(name + ".map")

print ("Map Saved Successfully")

elif event.key==pygame.K\_F10:

name = input("Map name:")

load\_map(name + ".map")

print ("Map Loaded Successfully!")

elif event.type == pygame.KEYUP:

camera\_move = 0

if event.type == pygame.MOUSEMOTION:

mouse\_pos = pygame.mouse.get\_pos()

mouse\_x = math.floor(mouse\_pos[0] / Tiles.Size) \* Tiles.Size

mouse\_y = math.floor(mouse\_pos[1] / Tiles.Size) \* Tiles.Size

if event.type == pygame.MOUSEBUTTONDOWN:

tile = [mouse\_x - camera\_x, mouse\_y - camera\_y, brush] #KEEP THIS AS A LIST

found = False

for t in tile\_data:

if t[0] == tile[0] and t[1] == tile[1]:

found = True

break

if not found:

if not brush == "r":

tile\_data.append(tile)

else:

if brush == "r":

#Remove Tile

for t in tile\_data:

if t[0] == tile[0] and t[1] == tile[1]:

tile\_data.remove(t)

print ("Tile Removed!")

else:

print ("A tile is already placed here!")

#LOGIC

if camera\_move == 1:

camera\_y += Tiles.Size

elif camera\_move == 2:

camera\_y -= Tiles.Size

elif camera\_move == 3:

camera\_x += Tiles.Size

elif camera\_move == 4:

camera\_x -= Tiles.Size

#Renbder Graphics

window.fill(Color.Blue)

#Draw Map

for tile in tile\_data:

try:

window.blit(Tiles.Texture\_Tags[tile[2]], (tile[0] + camera\_x, tile[1] + camera\_y))

except:

pass

#Draw Tile Highlighter (Selector)

window.blit(selector, (mouse\_x, mouse\_y))

pygame.display.update()

clock.tick(60)

pygame.quit()

sys.exit()

1. ***Interest.cpp***

#include <iostream>

#include "interest.h"

#include "IntAndDepClass.h"

using namespace std;

Interest::Interest(double p, double c, double m)

{

this->percent = p;

this->capital = c;

this->month = m;

}

double Interest::getSingleInterest ()const

{

double totalInt;

for (int i = 1; i <= month; i++)

{

totalInt = capital + capital \* percent/100 \* i;

cout << "In month " << i << ", your capital will be " << totalInt << endl;

}

return totalInt;

}

Interest::Interest(double p, double c, double m, int a )

{

this->percent = p;

this->capital = c;

this->month = m;

double \*ptr = &compoundInt;

ptr = new double[a];

for (int i = 0; i < a ; i++)

{

\*(ptr + i) = c\*pow((1+p/100), i+1);

cout << "In month " << i+1 << ", your capital will be " << ptr[i] << endl;

}

delete []ptr;

}

1. ***Globals.py***

class Globals:

camera\_x= 0

camera\_y= 0

camera\_move= 0

scene = "menu"

1. ***map\_engine.py***

import pygame

from scripts.textures import \*

class Map\_Engine:

def add\_tile(tile, pos, addTo):

addTo.blit(tile, (pos[0] \* Tiles.Size, pos[1] \* Tiles.Size))

def load\_map(file):

with open(file, "r") as mapfile:

map\_data = mapfile.read()

#Read Map Data

map\_data = map\_data.split("-") #Split into list of tiles

map\_size = map\_data[len(map\_data) - 1] #Get Map Dimensions

map\_data.remove(map\_size)

map\_size = map\_size.split(",")

map\_size[0] = int(map\_size[0]) \* Tiles.Size

map\_size[1] = int(map\_size[1]) \* Tiles.Size

tiles = []

for tile in range(len(map\_data)):

map\_data[tile] = map\_data[tile].replace("\n", "")

tiles.append(map\_data[tile].split(":")) #Split pos from texture

for tile in tiles:

tile[0] = tile[0].split(",") #Split pos into list

pos = tile[0]

for p in pos:

pos[pos.index(p)] = int(p) #Convert to integer

tiles[tiles.index(tile)] = (pos, tile[1]) #Save to tile list

#Create Terrain

terrain = pygame.Surface(map\_size, pygame.HWSURFACE)

for tile in tiles:

if tile[1] in Tiles.Texture\_Tags:

Map\_Engine.add\_tile(Tiles.Texture\_Tags[tile[1]], tile[0], terrain)

if tile[1] in Tiles.Blocked\_Types:

Tiles.Blocked.append(tile[0])

return terrain

1. ***meloonatic\_gui.py***

"""

Meloonatic Melons

GUI Framework

By Harry Hitchen

"""

import pygame

from scripts.UltraColor import \*

pygame.init()

def MouseOver(rect):

mouse\_pos = pygame.mouse.get\_pos()

if mouse\_pos[0] > rect[0] and mouse\_pos[0] < rect[0] + rect[2] and mouse\_pos[1] > rect[1] and mouse\_pos[1] < rect[1] + rect[3]:

return True

else:

return False

class Font:

Default = pygame.font.SysFont("Verdana", 20)

Small = pygame.font.SysFont("Verdana", 15)

Medium = pygame.font.SysFont("Verdana", 40)

Large = pygame.font.SysFont("Verdana", 60)

Scanner = pygame.font.SysFont("Verdana", 30)

class Menu:

class Button:

All = []

def \_\_init\_\_(self, text, rect, bg = Color.Gray, fg = Color.White, bgr = Color.CornflowerBlue, font = Font.Default, tag = ("menu", None)):

self.Text = text

self.Left = rect[0]

self.Top = rect[1]

self.Width = rect[2]

self.Height = rect[3]

self.Command = None

self.Rolling = False

self.Tag = tag

# NORMAL BUTTON

self.Normal = pygame.Surface((self.Width, self.Height), pygame.HWSURFACE|pygame.SRCALPHA)

self.Normal.fill(bg)

RText = font.render(text, True, fg) # text, antialiasing, color

txt\_rect = RText.get\_rect()

self.Normal.blit(RText, (self.Width / 2 - txt\_rect[2] / 2, self.Height / 2 - txt\_rect[3] / 2))

# HIGHLIGHTED BUTTON

self.High = pygame.Surface((self.Width, self.Height), pygame.HWSURFACE|pygame.SRCALPHA)

self.High.fill(bgr)

self.High.blit(RText, (self.Width / 2 - txt\_rect[2] / 2, self.Height / 2 - txt\_rect[3] / 2))

# SAVE BUTTON

Menu.Button.All.append(self)

def Render(self, to, pos = (0, 0)):

if MouseOver((self.Left + pos[0], self.Top + pos[1], self.Width, self.Height)):

to.blit(self.High, (self.Left + pos[0], self.Top + pos[1]))

self.Rolling = True

else:

to.blit(self.Normal, (self.Left + pos[0], self.Top + pos[1]))

self.Rolling = False

class Text:

All = []

def \_\_init\_\_(self, text, font = Font.Default, color = Color.Lime, bg = None):

self.Text = text

self.LastText = text

self.Font = font

self.Color = color

self.Left = 0

self.Top = 0

self.Bg = bg

bitmap = font.render(text, True, color)

self.Bitmap = pygame.Surface(bitmap.get\_size(), pygame.SRCALPHA|pygame.HWSURFACE)

if bg != None:

self.Bitmap.fill(bg)

self.Bitmap.blit(bitmap, (0, 0))

self.Width = self.Bitmap.get\_width()

self.Height = self.Bitmap.get\_height()

def Render(self, to, pos = (0, 0)):

if self.Text != self.LastText:

# TEXT HAS BEEN CHANGED

self.LastText = self.Text

# RECREATE BITMAP (Dynamic Text Rendering)

bitmap = self.Font.render(self.Text, True, self.Color)

self.Bitmap = pygame.Surface(bitmap.get\_size(), pygame.SRCALPHA|pygame.HWSURFACE)

if self.Bg != None:

self.Bitmap.fill(self.Bg)

self.Bitmap.blit(bitmap, (0, 0))

self.Width = self.Bitmap.get\_width()

self.Height = self.Bitmap.get\_height()

to.blit(self.Bitmap, (self.Left + pos[0], self.Top + pos[1]))

class Image:

def \_\_init\_\_(self, bitmap, pos = (0, 0)):

self.Bitmap = bitmap

self.Left = pos[0]

self.Top = pos[1]

self.Height = bitmap.get\_height()

self.Width = bitmap.get\_width()

def Render(self, to, pos = (0, 0)):

to.blit(self.Bitmap, (self.Left + pos[0], self.Top + pos[1]))

1. ***npc.py***

import pygame

pygame.init()

def get\_faces(sprite):

faces = {}

size = sprite.get\_size()

tile\_size = (int(size[0] / 2), int(size[1] / 2))

south = pygame.Surface(tile\_size, pygame.HWSURFACE|pygame.SRCALPHA)

south.blit(sprite, (0, 0), (0, 0, tile\_size[0], tile\_size[1]))

faces["south"] = south

north = pygame.Surface(tile\_size, pygame.HWSURFACE | pygame.SRCALPHA)

north.blit(sprite, (0, 0), (tile\_size[0], tile\_size[1], tile\_size[0], tile\_size[1]))

faces["north"] = north

east = pygame.Surface(tile\_size, pygame.HWSURFACE|pygame.SRCALPHA)

east.blit(sprite, (0, 0), (tile\_size[0], 0, tile\_size[0], tile\_size[1]))

faces["east"] = east

west = pygame.Surface(tile\_size, pygame.HWSURFACE | pygame.SRCALPHA)

west.blit(sprite, (0, 0), (0, tile\_size[1], tile\_size[0], tile\_size[1]))

faces["west"] = west

return faces

7.player.py

import pygame

from scripts.NPC import \*

pygame.init()

class Player:

def \_\_init\_\_(self, name):

self.name = name

self.facing = "south"

self.health = 100

sprite = pygame.image.load("graphics\\player.png")

size = sprite.get\_size()

self.width = size[0]

self.height = size[1]

#Get Player Faces

self.faces = get\_faces(sprite)

def render(self, surface, pos):

surface.blit(self.faces[self.facing], pos)

8.textures.py

import pygame

pygame.init()

class Tiles:

Size = 32

Blocked = []

Blocked\_Types = ["3"]

def Blocked\_At(pos):

if list(pos) in Tiles.Blocked:

return True

else:

return False

def Load\_Textures(file, Size):

bitmap = pygame.image.load(file)

bitmap = pygame.transform.scale(bitmap, (Size, Size))

surface = pygame.Surface((Size, Size), pygame.HWSURFACE|pygame.SRCALPHA)

surface.blit(bitmap,(0, 0))

return surface

Grass = Load\_Textures("graphics\\grass.png", Size)

Stone = Load\_Textures("graphics\\stone.png", Size)

Water = Load\_Textures("graphics\\water.png", Size)

Texture\_Tags = {"1" : Grass, "2" : Stone, "3" : Water}

9. UltraColor. """

UltraColor - Meloonatic Melons

Build for Python 3

Designed for PyGame

Version 1.2

http://www.youtube.com/meloonaticmelons

http://www.meloonaticmessage.btck.co.uk

"""

import random

class Color:

# Color Variables

# Transparent

Transparent = (0, 0, 0, 0)

Null = Transparent

Nothing = Transparent

Blank = Transparent

SeeThrough = Transparent

# White Shades

White = (255, 255, 255)

Silver = (150, 150, 150)

LightGray = (200, 200, 200)

Gainsboro = (220, 220, 220)

WhiteSmoke = (245, 245, 245)

# Black Shades

Black = (0, 0, 0)

Fog = (20, 20, 20)

DimGray = (50, 50, 50)

Gray = (70, 70, 70)

Mist = (120, 120, 120)

# Red Shades

Red = (255, 0, 0)

Blood = (30, 0, 0)

Maroon = (40, 0, 0)

DarkRed = (50, 0, 0)

Brown = (60, 50, 0)

RedBrown = (165, 42, 42)

Mud = (70, 60, 0)

Firebrick = (178, 34, 34)

IndianRed = (205, 92, 92)

LightCoral = (240, 128, 128)

RosyBrown = (188, 143, 143)

MistyRose = (255, 228, 225)

# Orange Shades

Orange = (255, 150, 0)

LightOrange = (255, 200, 0)

DarkOrange = (255, 100, 0)

Salmon = (250, 128, 114)

Tomato = (255, 99, 71)

DarkSalmon = (233, 150, 122)

Coral = (255, 127, 80)

OrangeRed = (255, 69, 0)

LightSalmon = (255, 160, 122)

Sienna = (160, 82, 45)

SeaShell = (255, 245, 238)

Chocolate = (210, 105, 30)

SaddleBrown = (139, 69, 19)

SandyBrown = (244, 164, 96)

PeachPuff = (255, 218, 185)

Peru = (205, 133, 63)

DarkOrange = (255, 140, 0)

BurlyWood = (222, 184, 135)

Tan = (210, 180, 140)

# Yellow Shades

Yellow = (255, 255, 0)

Linen = (250, 240, 230)

Bisque = (255, 228, 196)

AntiqueWhite = (250, 235, 215)

NavajoWhite = (255, 222, 173)

BlanchedAlmond = (255, 235, 205)

PapayaWhip = (255, 239, 213)

Moccasin = (255, 228, 181)

Wheat = (245, 222, 179)

OldLace = (253, 245, 230)

FloralWhite = (255, 250, 240)

DarkGoldenrod = (184, 134, 11)

Goldenrod = (218, 165, 32)

Cornsilk = (255, 248, 220)

Gold = (255, 215, 0)

Khaki = (240, 230, 140)

LemonChiffon = (255, 250, 205)

PaleGoldenrod = (238, 232, 170)

DarkKhaki = (189, 183, 107)

Beige = (245, 245, 220)

LightGoldenrodYellow = (250, 250, 210)

LightYellow = (255, 255, 224)

Ivory = (255, 255, 240)

# Green Shades

Green = (0, 128, 0)

Olive = (128, 128, 0)

OliveDrab = (107, 142, 35)

YellowGreen = (154, 205, 50)

DarkOliveGreen = (85, 107, 47)

GreenYellow = (173, 255, 47)

Chartreuse = (127, 255, 0)

LawnGreen = (124, 252, 0)

DarkSeaGreen = (143, 188, 139)

LightGreen = (144, 238, 144)

ForestGreen = (34, 139, 34)

LimeGreen = (50, 205, 50)

PaleGreen = (152, 251, 152)

DarkGreen = (0, 100, 0)

Lime = (0, 255, 0)

HoneyDew = (240, 255, 240)

SeaGreen = (46, 139, 87)

MediumSeaGreen = (60, 179, 113)

SpringGreen = (0, 255, 127)

MintCream = (245, 255, 250)

MediumSpringGreen = (0, 250, 154)

# Blue Shades

Blue = (0, 0, 255)

MediumAquamarine = (102, 205, 170)

Aquamarine = (127, 255, 212)

Turquoise = (64, 224, 208)

LightSeaGreen = (32, 178, 170)

MediumTurquoise = (72, 209, 204)

DarkSlateGray = (47, 79, 79)

PaleTurquoise = (175, 238, 238)

Teal = (0, 128, 128)

DarkCyan = (0, 139, 139)

Aqua = (0, 255, 250)

Cyan = (0, 255, 255)

LightCyan = (224, 255, 255)

Azure = (240, 255, 255)

DarkTurqoise = (0, 206, 209)

CadetBlue = (95, 158, 160)

PowderBlue = (176, 224, 230)

LightBlue = (173, 216, 230)

DeepSkyBlue = (0, 191, 255)

SkyBlue = (135, 206, 235)

LightSkyBlue = (135, 206, 250)

SteelBlue = (70, 130, 180)

AliceBlue = (240, 248, 255)

DodgerBlue = (30, 144, 255)

SlateGray = (112, 128, 144)

LightSlateGray = (119, 136, 153)

LightSteelBlue = (176, 196, 222)

CornflowerBlue = (100, 149, 237)

RoyalBlue = (65, 105, 225)

MidnightBlue = (25, 25, 112)

Lavender = (230, 230, 250)

Navy = (0, 0, 128)

DarkBlue = (0, 0, 139)

MediumBlue = (0, 0, 205)

GhostWhite = (0, 0, 205)

SlateBlue = (106, 90, 205)

# Purple Shades

Purple = (255, 0, 255)

DarkSlateBlue = (72, 61, 139)

MediumSlateBlue = (123, 104, 238)

MediumPurple = (147, 112, 219)

BlueViolet = (138, 43, 226)

Indigo = (75, 0, 130)

DarkOrchid = (153, 50, 204)

DarkViolet = (148, 0, 211)

MediumOrchid = (186, 85, 211)

Thistle = (216, 191, 216)

Plum = (221, 160, 221)

Violet = (238, 130, 238)

DarkMagenta = (139, 0, 139)

Orchid = (218, 112, 214)

MediumVioletRed = (199, 21, 133)

DeepPink = (255, 20, 147)

HotPink = (255, 105, 180)

LavenderBlush = (255, 240, 245)

PaleVioletRed = (219, 112, 147)

Crimson = (220, 20, 60)

Pink = (255, 192, 203)

LightPink = (255, 182, 193)

# Random Color Generator

def Random(includeAlpha=False):

if includeAlpha:

values = [0, 0, 0, 0]

else:

values = [0, 0, 0]

for c in range(len(values)):

values[c] = random.randrange(0, 255)

return tuple(values)

# Edit Transparency

def WithAlpha(alpha, color):

listColor = list(color)

listColor.append(alpha)

return tuple(listColor)py