ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

PROGRAMACIÓN AVANZADA

ASIGNATURA:	Programación Avanzada
PROFESOR:	Ing. Juan Pablo Zaldumbide
PERÍODO ACADÉMICO:	Octubre. 2016 - Marzo. 2017

REPORTE PROYECTO

TÍTULO:

AVANCE PERSONAL DEL PROYECTO FINAL

ESTUDIANTE

Edison Osorio

FECHA DE REALIZACIÓN: 29 de enero del 2017

FECHA DE ENTREGA: 01 de febrero del 2017

CALIFICACIÓN OBTENIDA:

1 MARCO TEÓRICO

1.1. Clases en Python

Es una mezcla de los mecanismos de clases encontrados en C++ y Modula-3. Las clases de Python proveen todas las características normales de la Programación Orientada a Objetos: el mecanismo de la herencia de clases permite múltiples clases base, una clase derivada puede sobrescribir cualquier método de su(s) clase(s) base, y un método puede llamar al método de la clase base con el mismo nombre. Los objetos pueden tener una cantidad arbitraria de datos de cualquier tipo. Igual que con los módulos, las clases participan de la naturaleza dinámica de Python: se crean en tiempo de ejecución, y pueden modificarse luego de la creación.

1.2. Parámetro self

Básicamente self sirve para que cada vez que se declara un método en Python, se va a tener que agregarle el argumento self para que cuando ese método sea invocado, Python le pase el objeto instanciado y así pueda operar con los valores actuales de esa instancia. Si no incluyes ese argumento y ejecutas el código Python disparará una excepción y el programa se detendrá.
El uso de self es la forma particular en que Python soporta los objetos, es como el this de C++, Java o PHP, self en Python no es una palabra reservada y se puede usar cualquier otra palabra, self es más una convención en el lenguaje que hace el código más legible, facilita el resaltado de sintaxis.

1.3. Sprite

La creación de sprites nos permitirá trabajar de una forma más cómoda con objetos que creemos en el videojuego ya que podremos acceder a propiedades muy interesantes y asimismo trabajar con programación orientada a objetos de manera más simple.
La clase Sprite está destinado a ser usado como una clase base para los distintos tipos de objetos en el juego. También hay una clase de grupo base que simplemente almacena los sprites. Un juego podría crear nuevos tipos de clases de grupo que operan en los casos Sprite especialmente diseñada que contienen.

2 DESARROLLO

2.1. Definición de los colores que van tener momentáneamente las paredes.

Para facilitar el uso de colores al momento de crear las paredes del laberinto, como del personaje principal entre otros objetos que vayamos a crear los declaramos como variables globales.

```
NEGRO = (0, 0, 0)
BLANCO = (255, 255, 255)
AZUL = (0, 0, 255)
VERDE = (0, 255, 0)
ROJO = (255, 0, 0)
VIOLETA = (255, 0, 255)
```

2.2. Creación de la clase pared

Creamos la clase pared que será la principal para crear todas las paredes del laberinto, las características que tendrán las paredes van como parámetros en la función que a su vez la llamamos como clase padre para poder llamar a métodos definidos en alguna de las clases de las que heredamos, y ponemos cuál será el origen de cada pared que se cree.

```
class Pared(pygame.sprite.Sprite):
    def __init__(self, x, y, largo, alto, color):

        super().__init__()

        self.image = pygame.Surface([largo, alto])
        self.image.fill(color)

        self.rect = self.image.get_rect()
        self.rect.y = y
        self.rect.x = x
```

2.3. Creación de la clase cuarto

Creamos esta clase para que sea la clase base para almacenar las paredes que crearemos más adelante y los enemigos que existirán en cada cuarto o fase.

```
class Cuarto():

    pared_lista = None
    sprites_enemigos = None

    def __init__(self):

        self.pared_lista = pygame.sprite.Group()
        self.sprites_enemigos = pygame.sprite.Group()
```

2.4. Creación de las fases (cuartos) del juego

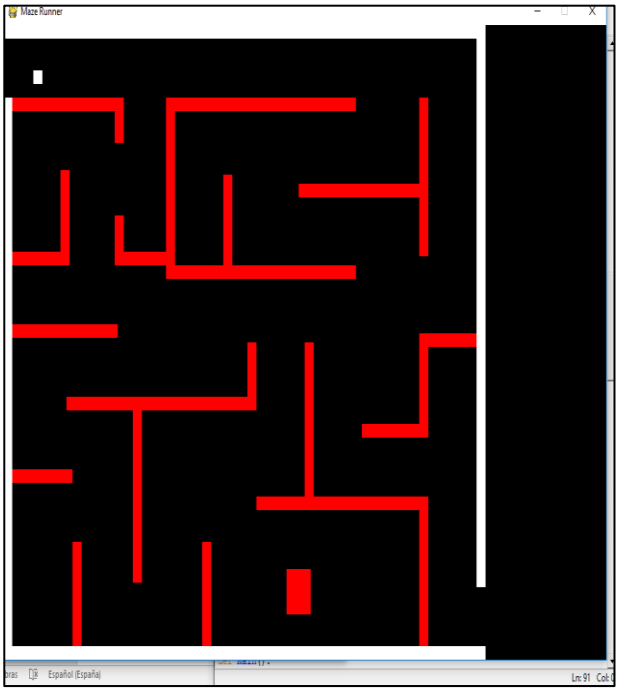
En esta clase ya creamos lo que son las paredes del laberinto para lo cual llamamos a la función padre para poder heredar los atributos antes declarados.

```
class Cuarto1(Cuarto):

    def __init__(self):
        super().__init__()
```

```
        paredes = [ [0,80,15,620,BLANCO],#|
                  [785,0,15,620,BLANCO],#|#
                  [0,0,785,15,BLANCO],
                  [15,685,785,15,BLANCO],
                  [15,80,175,15,ROJO],
                  [185,80,15,50,ROJO],
                  [95,160,15,90,ROJO],
                  [15,250,95,15,ROJO],
                  [15,330,175,15,ROJO],
                  [185,210,15,55,ROJO],
                  [185,250,85,15,ROJO],
                  [270,80,15,200,ROJO],
                  [285,80,300,15,ROJO],
                  [285,265,300,15,ROJO],
                  [365,165,15,100,ROJO],
                  [690,80,15,175,ROJO],
                  [490,175,200,15,ROJO],
                  [420,520,270,15,ROJO],
                  [690,520,15,165,ROJO],
                  [690,340,15,100,ROJO],
                  [690,340,95,15,ROJO],
                  [595,440,110,15,ROJO],
                  [215,410,15,205,ROJO],
                  [105,410,300,15,ROJO],
                  [15,490,100,15,ROJO],
                  [115,570,15,115,ROJO],
                  [330,570,15,115,ROJO],
                  [470,600,40,50,ROJO],#enemigo
                  [405,350,15,75,ROJO],
                  [500,350,15,170,ROJO],
                ]

        for item in paredes:
            pared = Pared(item[0],item[1],item[2],item[3],item[4])
            self.pared_lista.add(pared)
```



En el for al final de la clase, lo creamos para ir almacenando todas las paredes y después junto con el cuarto llamarlos al pasar de fase.

2.5. En el main

En la función principal creamos un arreglo donde iremos almacenando las fases para luego, mediante la posición en la que están almacenadas las fases, poder ir cambiando.

```
cuartos = []

cuarto = Cuarto1()
cuartos.append(cuarto)

cuarto = Cuarto1()
cuartos.append(cuarto)

cuarto_actual_no = 0
cuarto_actual = cuartos[cuarto_actual_no]
```

2.6. Cambio de Fase

Con lo hecho anterior mente y con if podremos ir cambiando de fase.

```
protagonista.mover(cuarto_actual.pared_lista)

if protagonista.rect.x < -15:
    if cuarto_actual_no == 0:
        cuarto_actual_no = 2
        cuarto_actual = cuartos[cuarto_actual_no]

    elif cuarto_actual_no == 2:
        cuarto_actual_no = 1
        cuarto_actual = cuartos[cuarto_actual_no]
```

3 PROBLEMAS ENCONTRADOS

- El primer problema que nos surgió es haber hecho partes del código por separado, ya que uno de los integrantes estaba trabajando solo con Tkinter y al tratar de unir el código se creó un conflicto con lo realizado con pygame, por lo que se tuvo que rehacer el trabajo.
- En lo que se refiere a la codificación se tuvo graves inconvenientes para adaptarse a funciones de pygame, ya que no se había utilizado antes por lo que no se sabía que funciones nos servirían para hacer lo planeado
- El problema más grande que hemos tenido y tenemos hasta ahora es la incorporación de una imagen con su spriter al laberinto, por lo que hasta el momento solo se ha dibujado al "personaje principal"
- La parte de los disparos que se quiere incorporar es otro problema que se nos ha presentado hasta el momento, aun no se sabe si llegue a ser complicado ya que aún no se ha programado ciento por ciento esta parte.

4 CONCLUSIONES

- Luego de una ardua investigación sobre lo básico de la librería de pygame se logró empezar a desarrollar el proyecto, además de obtener conocimiento imprevisto sobre otras librerías que se usarán en la culminación del proyecto.
- Se logró superar gran cantidad de inconvenientes presentados tanto en la codificación como en el aspecto grupal.

5 ORGANIZACIÓN DE LAS ACTIVIDADES

En general el trabajo realizado hasta la fecha se distribuyó de la siguiente manera:

Actividad\Integrantes	Danilo Benavidez	Michael Cárdenas	Edison Osorio
Obtención herramientas para el desarrollo del proyecto	✓	✓	✓
Elaboración de laberinto			✓
Creación del personaje	✓		
Creación de niveles			✓
Realización del proceso de movimiento del personaje	✓		
Elaboración del menú inicial		✓	
Puntaje, reloj		✓	

6 OPNINON

- ✓ Crear un juego es muy interesante y entretenido, lo único malo es no haber tenido la introducción necesaria en el lenguaje en el que se está creando y en las funciones básicas de la biblioteca que se usa como referencia principal.

7 BIBLIOGRAFÍA

1. [1]2017. [Online]. Available: http://2014.es.pycon.org/static/talks/Clases%20en%20Python,%20Lo%20est%C3%A1s%20haciendo%20mal%20-%20V%C3%ADctor%20Terr%C3%B3n.pdf. [Accessed: 29- Jan- 2017].
2. [2]P. Craven, "Programar Juegos Arcade con Python y Pygame", *Programarcadegames.com*, 2017. [Online]. Available: http://programarcadegames.com/index.php?chapter=introduction_to_graphics&lang=es. [Accessed: 25- Jan- 2017].

3. [3]"pygame.sprite — Pygame v1.9.2 documentation", *Pygame.org*, 2017. [Online]. Available: https://www.pygame.org/docs/ref/sprite.html. [Accessed: 30- Jan- 2017].
4. [4]"Foros del Web", *Forosdelweb.com*, 2017. [Online]. Available: http://www.forosdelweb.com/f130/parametro-self-python-dudas-964196/. [Accessed: 30- Jan- 2017].