PROYECTO FINAL

“SPACE INVADERS”

AUTOR (ES)

- **CORO ANDRÉS**
- **IZA JORGE**
- **TIPÁN JENNY**

ESCUELA POLITÉCNICA NACIONAL

TECNOLOGÍA EN ANÁLISIS DE SISTEMAS INFORMÁTICOS

Jueves 15 de febrero del 2018

Quito – Ecuador

2017 – 2018

ÍNDICE GENERAL

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

INFORME FINAL PROYECTO

JUEGO SPACE INVADERS

CORO ANDRÉS

IZA JORGE

TIPÁN JENNY

ESCUELA POLITÉCNICA NACIONAL DEL ECUADOR

TECNOLOGÍA EN ANÁLISIS DE SISTEMAS INFORMÁTICOS

1. INTRODUCCIÓN

El presente proyecto tiene como fin presentar la implementación de un juego tipo arcade realizado en Python, dentro de este se detallará el proceso realizado y las herramientas utilizadas para desarrollar el programa. Es importante conocer versiones de programas estables y compatibles con el equipo y con los demás programas que se vayan a utilizar antes de iniciar el desarrollo de cualquier juego.

2. OBJETIVO GENERAL

- *Implementar un juego tipo arcade en Python utilizando todas las herramientas vistas en clase y aplicar los conocimientos adquiridos en el presente semestre.*

3. OBJETIVOS ESPECÍFICOS

- *Descargar todos los programas necesarios para la implementación del juego.*
- *Descargar imágenes y sonidos necesarios para el desarrollo del programa.*
- *Implementar detección de sonidos, reloj, puntajes y botones.*
- *Realizar un archivo ejecutable del juego.*

4. MARCO TEÓRICO

Para el desarrollo del programa se usó Python que es un lenguaje independiente de plataforma y orientado a objetos, puede realizar cualquier tipo de programa. Es un lenguaje interpretado, es decir que no necesita compilar el código fuente para poder ejecutarlo, una de sus ventajas es la rapidez de desarrollo e inconvenientes como una menor velocidad. [1]

El desarrollo del programa lo hemos realizado en Pycharm que es un IDE muy completo. Este IDE es profesional y viene en dos modalidades: una edición Free y otra muy completa privada que apunta a empresas de desarrollo de software. Características como desarrollo remoto, soporte de bases de datos, soporte de desarrollo web, etc, están disponibles solo para la edición profesional de PyCharm. [2]

La utilidad de Pycharm es su compatibilidad con múltiples marcos de desarrollo web de terceros lo que lo convierte en un completo IDE de desarrollo de aplicaciones rápidas. [2]

Para una mejor visualización de nuestro programa usamos Pygame que es un conjunto de módulos del lenguaje Python que permiten la creación de videojuegos en dos dimensiones de una manera sencilla. Está orientado al manejo de sprites. También puede utilizarse para crear otros programas multimedia o interfaces gráficas de usuario. Pygame está basado en la librería SDL 1.2, una alternativa más actual de SDL en Python podría ser Py-SDL2, que implementa varias mejoras respecto a Pygame. [3]

Funciona como interfaz de las bibliotecas SDL. Las SDL (Simple DirectMedia Layer) son un conjunto de bibliotecas que proporcionan funciones básicas para realizar operaciones de dibujado 2D, gestión de efectos de sonido,

música, carga y gestión de imágenes (y son importantes ya que pygame trabaja sobre ellas). [3]

Esta librería es muy útil con el manejo de imágenes ya que se puede cargar una con todas las imágenes que se desee y manejar estas a través de sus coordenadas de posición, y no cargar varias imágenes.

Para convertir nuestro archivo de Python con toda la implementación realizada en un archivo ejecutable usamos cx_Freeze que es un conjunto de scripts y módulos para congelar scripts de Python en ejecutables de la misma manera que py2exe y py2app. A diferencia de estas dos herramientas, cx_Freeze es multiplataforma y debería funcionar en cualquier plataforma que Python en sí mismo funciona. Requiere Python 2.7 o superior y funciona con Python 3. [4]

Además, utilizamos la aplicación SourceTree es quizás uno de los mejores clientes GUI para manejar repositorios git que existen en la actualidad. [5]

A través de esta herramienta se puede realizar completamente todas las tareas de gestión de un DVCS: [5]

- *Crear y clonar repositorios de cualquier sitio*
- *Detectar y resolver conflictos*
- *Consultar el historial de cambios de nuestros repositorios.*

Este programa los usamos junto con una cuenta GitHub, para todos los miembros del grupo disponer del archivo que vamos a realizar. [5]

5.1. IMPLEMENTACIÓN PROGRAMA

Ahora vamos a indicar como hemos implementado nuestro programa.

El juego que decidimos implementar es Space Invaders:

Lo primero que debemos hacer es cargar las imágenes que se van a utilizar en la carpeta en la que se encuentra nuestro archivo .py.

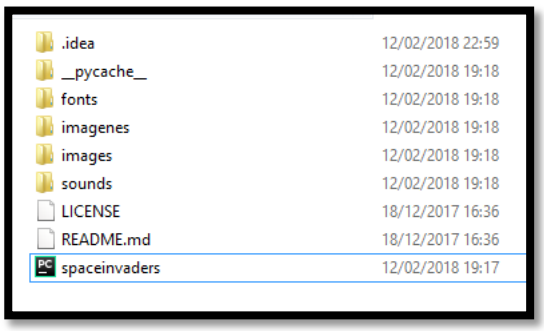

Ilustración 1 Carga de imágenes y sonidos

En el código lo primero que realizamos es la importación de librerias.

pygame: *librería utilizada para la creación de videojuegos, uso de sprites,etc.*

sys: *Este módulo proporciona acceso a algunas variables utilizadas o mantenidas por el intérprete y a funciones que interactúan fuertemente con el intérprete.*

shuffle*: módulo utilizado para mezclar una lista de objetos*

choice*: devuelve un elemento aleatorio de una lista, tupla o cadena.*

random: *para generar valores aleatorios*

```
from pygame import *
import sys
from random import shuffle, randrange, choice
```

Ilustración 2 Implementación Código librerías

Lo siguiente que realizamos es definir los colores que vamos a utilizar, los colores están en formato RGB.

Luego establecemos los datos globales para la creación de ventanas. Además, el programa permitirá cualquier tipo de letra.

Se carga todas las imágenes que se van a utilizar en formato .png. Luego pasamos a convertir estas imágenes a un formato adecuado.

```
#todo     R    G    B
WHITE = (255, 255, 255)
GREEN = (78, 255, 87)
YELLOW = (241, 255, 0)
BLUE = (80, 255, 239)
PURPLE = (203, 0, 255)
RED = (237, 28, 36)

SCREEN = display.set_mode((800, 650)) #todo tamaño de nuestra ventana de
FONT = "fonts/space_invaders.ttf" #todo tipo de letra
IMG_NAMES = ["ship", "enemy1_1", "enemy1_2", "enemy2_1", "enemy2_2", "ene
             "explosionblue", "explosiongreen", "explosionpurple", "laser
IMAGES = {name: image.load("images/{}.png".format(name)).convert_alpha()
          for name in IMG_NAMES} #todo llamamos a las imagenes del anteri
```

Ilustración 3 Implementación Código colores e imágenes

Seguimos con la ceración de clases.

En la clase NaveEspacial se implementó dos funciones, en la primera función se carga de imagen de la nave, ajustada en un determinado tamaño y se define su velocidad.

En la segunda función establecemos el movimiento de las imágenes con las teclas de dirección y se establece los limites hasta dónde puede llegar la imagen.

La imagen se carga en la pantalla game.

```
class calseNaveEspacial(sprite.Sprite):
    def __init__(self):
        sprite.Sprite.__init__(self)  #todo inicializamos los sprite
        self.ImagenNave = image.load("imagenes/nave.png")  #todo carg
        self.ImagenNave = transform.scale(self.ImagenNave, (90, 90))
        self.rect = self.ImagenNave.get_rect(topleft=(375, 540))  #to
        self.speed = 5  #todo velocidad de la nave

    def update(self, keys, *args):
        if keys[K_LEFT] and self.rect.x > 10:#todo damos el limite ha
            self.rect.x -= self.speed#todo mov de la nave a la IZQ y
        if keys[K_RIGHT] and self.rect.x < 740:#todo limite de la der
            self.rect.x += self.speed#todo mov de la nave a la derech
        game.screen.blit(self.ImagenNave, self.rect)
```

Ilustración 4 Implementación Código Clase Nave-Espacial

La clase Proyectil tiene implementada dos funciones.

La primera función recibe como parámetros las coordenadas x, y, la velocidad, nombre del archivo, tamaño, todos los datos son para el proyectil. Se carga la imagen, se define el tamaño de esta y la velocidad que tendrá

La segunda función carga la imagen en la pantalla game, y se define hasta donde deben llegar los proyectiles, además está dada la condición para que indique cuando toco a un enemigo.

```
class claseProyectil(sprite.Sprite):
    def __init__(self, xpos, ypos, direction, speed, filename, side): #todo
        sprite.Sprite.__init__(self)
        self.image = IMAGES[filename]
        self.rect = self.image.get_rect(topleft=(xpos, ypos))#todo tamaño del
        self.speed = speed#todo velocidad del proyectil
        self.direction = direction
        self.side = side
        self.filename = filename

    def update(self, keys, *args):
        game.screen.blit(self.image, self.rect)
        self.rect.y += self.speed * self.direction
        print(self.rect.y)
        if self.rect.y < 15 or self.rect.y > 600:
            self.kill()
```

Ilustración 5 Implementación Clase Proyectil

La clase Invasor tiene tres funciones implementadas.

La primera función contiene todos los atributos que se la darán a las imágenes de los enemigos, los movimientos que realizaran, las columnas que formaran, cuantos enemigos se cargaran en cada fila y cada que tiempo se irán desplazando hacia abajo.

```
class claseInvasor(sprite.Sprite):
    def __init__(self, row, column):
        sprite.Sprite.__init__(self)

        #todo atributos
        self.row = row
        self.column = column
        self.images = []
        self.load_images()
        self.index = 0
        self.image = self.images[self.index]
        self.rect = self.image.get_rect()
        self.direction = 1
        self.rightMoves = 15
        self.leftMoves = 30
        self.moveNumber = 0
        self.moveTime = 600
        self.firstTime = True
        self.movedY = False;
        self.columns = [False] * 10
        self.aliveColumns = [True] * 10
        self.addRightMoves = False
        self.addLeftMoves = False
        self.numOfRightMoves = 0
        self.numOfLeftMoves = 0
        self.timer = time.get_ticks()
```

Ilustración 6 Implementación Clase Invasor

La segunda función tiene implementado todas las condiciones de movimiento, es decir cuando las imágenes se mueven hacia la derecha, cuando se mueven hacia la izquierda, cuando están en el borde derecho y deban bajar, lo mismo cuando están en el borde de la izquierda y deben

baja, además está el control de velocidad de movimiento de las imágenes.

```
def update(self, keys, currentTime):

    if currentTime - self.timer > self.moveTime:
        self.movedY = False;
        #todo mov en x (der) y baja
        if self.moveNumber >= self.rightMoves and self.direction == 1:
            self.direction *= -1
            self.moveNumber = 0
            self.rect.y += 35
            self.movedY = True
            if self.addRightMoves:
                self.rightMoves += self.numOfRightMoves
            if self.firstTime:
                self.rightMoves = self.leftMoves;
                self.firstTime = False;
            self.addRightMovesAfterDrop = False
        #todo mov en x (izq) controla que baje(-1)
        if self.moveNumber >= self.leftMoves and self.direction == -1:
            self.direction *= -1
            self.moveNumber = 0
            self.rect.y += 35
            self.movedY = True
            if self.addLeftMoves:
                self.leftMoves += self.numOfLeftMoves
            self.addLeftMovesAfterDrop = False
        #todo mov en x para que empiece el movimiento
        if self.moveNumber < self.rightMoves and self.direction == 1 and not self.movedY:
            self.rect.x += 10
            self.moveNumber += 1
        #todo cuando llega al tope de mov en x (der)y baja(-1)
        if self.moveNumber < self.leftMoves and self.direction == -1 and not self.movedY:
            self.rect.x -= 10
            self.moveNumber += 1

        self.index += 1
        #todo controla la lista de los invasores para cargar y no se desvorde
        if self.index >= len(self.images):
            self.index = 0#todo controla que no se desvorde las imagenes de la lista

        #todo controla la velocidad en que bajan los invasores
        self.image = self.images[self.index]

        self.timer += self.moveTime
    game.screen.blit(self.image, self.rect)
```

Ilustración 7 Implementación Código Clase Invasor

En la tercera función se establece que imágenes van a estar en cada fila, el número de imágenes que se van a cargar y la escala adecuada que tendrán.

```
def load_images(self):
    images = {0: ["1_2", "1_1"],
              1: ["2_2", "2_1"],
              2: ["2_2", "2_1"],
              3: ["3_1", "3_2"],
              4: ["3_1", "3_2"],
              }
    img1, img2 = (IMAGES["enemy{}".format(img_num)] for img_num in images[self.row])
    self.images.append(transform.scale(img1, (40, 35)))
    self.images.append(transform.scale(img2, (40, 35)))
```

Ilustración 8 Implementación Clase Invasor

La clase Explosion tiene implementada tres funciones.

En la primera función se estable la carga de la imagen nave, con su respectiva escala

Esta función detectara el disparo del enemigo que impactara en la nave, la imagen del láser estará ajustada con su respectiva escala y que se cargara en la pantalla.

```
class Explosion(sprite.Sprite):
    def __init__(self, xpos, ypos, row, ship):
        sprite.Sprite.__init__(self)
        self.isShip = ship
        if ship:
            self.ImagenNave = image.load("imagenes/nave.png")  #todo carga
            self.ImagenNave = transform.scale(self.ImagenNave, (80, 80))
            self.rect = self.ImagenNave.get_rect(topleft=(xpos, ypos))

        else:#todo efecto de la explosion del invasor
            self.row = row
            self.load_image()
            self.image = transform.scale(self.image, (40, 35)) #todo escal
            self.rect = self.image.get_rect(topleft=(xpos, ypos))
            game.screen.blit(self.image, self.rect)

        self.timer = time.get_ticks()
```

Ilustración 9 Implementación Clase Explosión Función 1

La segunda función detectará el disparo que da el enemigo, la imagen se desvanece un instante y luego aparece en el centro de la pantalla.

La tercera función determina los colores de la explosión que se provoca cuando un disparo llega a la nave.

```
def update(self, keys, tiempoActual):
    if self.isShip:
        if tiempoActual - self.timer > 300 and tiempoActual - self.timer <= 600:
            game.screen.blit(self.ImagenNave, self.rect)
        if tiempoActual - self.timer > 900:
            self.kill()  #todo remueve el sprite del grupo
    else: #todo el tiempo que le toma en desaparecer el efecto de explosion del in
        if tiempoActual - self.timer <= 100:
            game.screen.blit(self.image, self.rect)
        if tiempoActual - self.timer > 100 and tiempoActual - self.timer <= 200:
            self.image = transform.scale(self.image, (50, 45))
            game.screen.blit(self.image, (self.rect.x - 6, self.rect.y - 6))
        if tiempoActual - self.timer > 400:
            self.kill()  #todo remueve el sprite del grupo

# todo metodo para cargar la imagen del invasor cuando explota
def load_image(self):
    imgColors = ["purple", "blue", "blue", "green", "green"]
    self.image = IMAGES["explosion{}".format(imgColors[self.row])]
```

Ilustración 10 Implementación Clase Explosión Función 2 y Función 3

La clase Vida tienen implementada dos funciones en la primera función se hace el llamado a la imagen nave, se establece la escala que tendrá.

La función dos lo que hace es cargar la imagen en la pantalla.

```python
class claseVida(sprite.Sprite):
    def __init__(self, xpos, ypos):
        sprite.Sprite.__init__(self)
        self.image = IMAGES["ship"]
        self.image = transform.scale(self.image, (23, 23))
        self.rect = self.image.get_rect(topleft=(xpos, ypos))

    def update(self, keys, *args):
        game.screen.blit(self.image, self.rect)
```

Ilustración 11Implementación Clase Vida

La clase Text tiene implementado dos funciones la primera función hace el llamado al tipo de letra que tendrá, que como antes se mencionó el programa podrá utilizar cualquier tipo de letra, el color que tendrá, y la posición en la que se imprimirá.

La segunda función hace la carga del texto en la pantalla.

```python
class Text(object):
    def __init__(self, textFont, size, message, color, xpos, ypos):
        self.font = font.Font(textFont, size)
        self.surface = self.font.render(message, True, color)
        self.rect = self.surface.get_rect(topleft=(xpos, ypos))

    def draw(self, surface):
        surface.blit(self.surface, self.rect)
```

Ilustración 12 Implementación Clase Text

La clase SpaceInvaders es la que tiene la implementación del desarrollo del juego, en si en esta clase están todos los datos del juego.

En la primera función se establece el nombre que tienen la venta que es Space Invaders, se carga la imagen de fondo del juego, como se sabe el juego debe reiniciar cuando el usuario salió del juego, entonces aquí se da valores booleanos a inicio del juego fin del juego, la posición de inicio de los enemigos para la partida actual y un contador de enemigos que ira aumentado mientras se siga avanzando en el juego.

```python
class SpaceInvaders(object):
    def __init__(self):
        mixer.pre_init(44100, -16, 1, 512)
        init()
        self.caption = display.set_caption('Space Invaders')
        self.screen = SCREEN
        self.background = image.load('imagenes/espacio.png').convert()
        self.startGame = False
        self.mainScreen = True  #todo  verdadero para que desaparesca de
        self.gameOver = False
        self.enemyPositionDefault = 65  #todo  inicial valores para un nu
        self.enemyPositionStart = self.enemyPositionDefault  # todo Conta
        self.enemyPosition = self.enemyPositionStart  # todo Posición de
```

Ilustración 13 Implementación Clase SpaceInvaders Función 1

La segunda función es usada para resetear el juego, es decir una vez que se cierra el juego, volvemos a la pantalla de menú, al decidir jugar nuevamente es necesario resetear el juego, una vez que se hace eso se requiere volver a cargar las imágenes de enemigos, naves, vidas, tiempo, pero esto solo se usa para un nuevo juego, no para una nueva partida.

```python
    def reset(self, score, lives, newGame=False):
        self.player = calseNaveEspacial()
        self.playerGroup = sprite.Group(self.player)
        self.explosionsGroup = sprite.Group()
        self.bullets = sprite.Group()
        self.enemyBullets = sprite.Group()
        self.reset_lives(lives)
        self.enemyPosition = self.enemyPositionStart
        self.make_enemies()
        # todo Solo crea bloqueadores en un juego nuevo, no e
        if newGame:
            self.keys = key.get_pressed()
            self.clock = time.Clock()
            self.timer = time.get_ticks()
            self.noteTimer = time.get_ticks()
            self.shipTimer = time.get_ticks()
            self.score = score
            self.lives = lives
            self.create_audio()
            self.create_text()
            self.killedRow = -1
            # self.killedColumn = -1
            self.makeNewShip = False
            self.shipAlive = True
            # self.killedArray = [[0] * 10 for x in range(5)]
```

Ilustración 14 Implementación Clase SpaceInvaders Función 2

En la tercera función se resetea la imagen de las vidas, cada vez que la nave recibe un disparo.

La cuarta función es usada para establecer los sonidos que tendrá el juego, habrá un sonido para el juego en general, sonidos para cada disparo, para colisiones de disparos de enemigos hacia la nave y viceversa, para los sonidos se establece el volumen que tendrán. Además, es importante saber que el juego mientras vaya avanzando la

música deberá seguir sonando por ello se establece las veces de repetición que tendrá.

```
def reset_lives(self, lives):
    self.lives = lives
    self.reset_lives_sprites()

def create_audio(self):
    self.sounds = {}
    for sound_name in ["shoot", "shoot2", "invaderkilled", "shipexplosion"]:
        self.sounds[sound_name] = mixer.Sound("sounds/{}.wav".format(sound_name))
        self.sounds[sound_name].set_volume(0.2)

    self.musicNotes = [mixer.Sound("sounds/{}.wav".format(i)) for i in range(4)]
    for sound in self.musicNotes:
        sound.set_volume(0.5)

    self.noteIndex = 0
```

Ilustración 15 Implementación Clase SpaceInvaders Función 3 y Función 4

La quinta función sirve para dar play a los sonidos dependiendo de los eventos que se dan, es decir que cuando se dispara tendrá un sonido que se escuchará en el momento que se dispara, cuando hay un impacto entre láser y nave y enemigo y láser también tendrá un sonido que de la misma manera solo se escuchar el momento que tengan impacto.

```
def play_main_music(self, currentTime):
    moveTime = self.enemies.sprites()[0].moveTime
    if currentTime - self.noteTimer > moveTime:
        self.note = self.musicNotes[self.noteIndex]
        if self.noteIndex < 3:
            self.noteIndex += 1
        else:
            self.noteIndex = 0

        self.note.play()
        self.noteTimer += moveTime
```

Ilustración 16 Implementación Clase SpaceInvaders Función 5

La sexta función tiene la implementación de los eventos que se realizan por teclado, hay condiciones si se presiona salir entonces saldremos del juego, pero si se dispara con la tecla espacio se genera otras condiciones se carga la imagen laser por cada disparo, se escucha el sonido del laser cuando el score definido es superado la nave dispara otro tipo de láser.

```
def check_input(self):
    self.keys = key.get_pressed()
    for e in event.get():
        if e.type == QUIT:
            sys.exit()
        if e.type == KEYDOWN:
            if e.key == K_SPACE:
                if len(self.bullets) == 0 and self.shipAlive:
                    if self.score < 500: #todo disparos con un solo laser si el score es menor a 500 pts
                        bullet = claseProyectil(self.player.rect.x + 23, self.player.rect.y + 5, -1, 15, "laser","center")
                        self.bullets.add(bullet)
                        self.allSprites.add(self.bullets)
                        self.sounds["shoot"].play()
                    else: #todo doble laser cuando el score pase los 500 pts
                        leftbullet = claseProyectil(self.player.rect.x + 8, self.player.rect.y + 5, -1, 15, "laser",
                                                    "left")
                        rightbullet = claseProyectil(self.player.rect.x + 38, self.player.rect.y + 5, -1, 15,
                                                     "laser", "right")

                        self.bullets.add(leftbullet)
                        self.bullets.add(rightbullet)
                        self.allSprites.add(self.bullets)
                        self.sounds["shoot2"].play()
```

Ilustración 17 Implementación Clase SpaceInvaders Función 6

La séptima función nos permite crear todos los textos que son necesarios para el juego. Los datos que son para la presentación de la pantalla y los que son para la pantalla de juego. Aquí se define la posición en la que se imprimirán.

```
def create_text(self):

    self.titulo_programacion = Text(FONT, 40, "PROGRAMACION   AVANZADA", BLUE, 40, 40)  # X,Y
    self.titulo_autores1 = Text(FONT, 20, "BY: IZA JORGE", BLUE, 550, 500)
    self.titulo_autores2 = Text(FONT, 20, "TIPAN JENNY", BLUE, 590, 540)
    self.titulo_autores3 = Text(FONT, 20, "CORO ANDRES", BLUE, 590, 580)
    self.titulo_SpaceInvader = Text(FONT, 60, "Space  Invaders", YELLOW, 140, 200)
    self.titulo_pres_tecla = Text(FONT, 25, "Press  any  key  to  continue", GREEN, 201, 350)
    self.titulo_boton_inicio = Text(FONT, 25, "Star Game:", RED, 200, 410)
    self.titulo_boton_salir = Text(FONT, 25, "Exit:", RED, 300, 510)
    self.gameOverText = Text(FONT, 50, "Game Over", WHITE, 250, 270)
    self.nextRoundText = Text(FONT, 50, "Next Round", WHITE, 240, 280)
    self.scoreText = Text(FONT, 20, "Score", WHITE, 5, 5)
    self.livesText = Text(FONT, 20, "Lives ", WHITE, 640, 5)
```

Ilustración 18 Implementación Clase SpaceInvaders Función 7

La octava función sirve para crear las columnas de enemigos y se define la posición que tendrán, aquí se establece el número de columnas que habrá y cuantas filas se generan.

```
def make_enemies(self):
    enemies = sprite.Group()
    for row in range(5):
        for column in range(10):
            enemy = claseInvasor(row, column)
            enemy.rect.x = 157 + (column * 50)
            enemy.rect.y = self.enemyPosition + (row * 45)
            enemies.add(enemy)

    self.enemies = enemies
    self.allSprites = sprite.Group(self.player, self.enemies, self.livesGroup)
```

Ilustración 19 Implementación Clase SpaceInvaders Función 8

La novena función sirve para generar los disparos de los enemigos, los enemigos van a estar en filas, solo disparara el que este primero en cada columna, es decir que al ser disparado un enemigo este desaparecerá entonces disparar el siguiente que está en la columna. Los disparos

siempre saldrán del centro del enemigo. Y tendrán un sonido diferente al de la nave.

```
def make_enemies_shoot(self):
    columnList = []
    for enemy in self.enemies:
        columnList.append(enemy.column)

    columnSet = set(columnList)
    columnList = list(columnSet)
    shuffle(columnList)
    column = columnList[0]
    enemyList = []
    rowList = []

    for enemy in self.enemies:
        if enemy.column == column:
            rowList.append(enemy.row)
    row = max(rowList)
    for enemy in self.enemies:
        if enemy.column == column and enemy.row == row:
            if (time.get_ticks() - self.timer) > 700:
                self.enemyBullets.add(
                    claseProyectil(enemy.rect.x + 14, enemy.rect.y + 20, 1, 5, "enemylaser", "center"))
                self.allSprites.add(self.enemyBullets)
                self.timer = time.get_ticks()
```

Ilustración 20 Implementación Clase SpaceInvaders Función 9

La décima función sirve para determinar el valor que representa cada imagen de los enemigos y este puntaje se ira acumulando de acuerdo con los enemigos que se haya disparado.

```
def calculate_score(self, row):
    scores = {0: 30,
              1: 20,
              2: 20,
              3: 10,
              4: 10,
              5: choice([50, 100, 150, 300])
              }

    score = scores[row]
    self.score += score
    return score
```

Ilustración 21 Implementación Clase SpaceInvaders Función 10

La función crear menú tiene la implementación cargar imágenes que estarna en esta pantalla, se establece la escala de las imágenes, se crea los botones y se establece los eventos que se realizarán con el mouse, se condiciona la detección del mouse a la posición de los botones, cuando se seleccione el botón "Start Game" se direcciona a la ventana inicio de juego y si se selecciona la posición del botón "Exit" el cual sale del programa.

```
def create_main_menu(self):
    self.ImagenEpn = image.load("imagenes/escudoEpn.png")  #todo carga
    self.ImagenEpn = transform.scale(self.ImagenEpn, (150, 150))  #tod
    self.ImagenBoton = image.load("imagenes/navePintada.png")
    self.ImagenBoton = transform.scale(self.ImagenBoton, (50, 50))  #t

    self.screen.blit(self.ImagenEpn, (650, 20))  # todo crea el espaci
    self.screen.blit(self.ImagenBoton, (400, 400))  #todo  boton star
    self.screen.blit(self.ImagenBoton, (400, 500))

    # evento del mouse clic
    for e in event.get():
        if e.type == QUIT:
            sys.exit()
        if e.type == MOUSEBUTTONDOWN:
            x, y = mouse.get_pos()
            print("evento mouse X" + str(x) + " y: " + str(y))
            if (x >= 400 and x <= 440 and y >= 400 and y <= 440):
                self.startGame = True
                self.mainScreen = False
            if (x >= 400 and x <= 440 and y >= 500 and y <= 540):
                sys.exit()
```

Ilustración 22 Implementación Clase SpaceInvaders Función 11

Esta función detecta la colisión de eventos primero detectamos los impactos de los disparos de la nave hacia los enemigos, se detecta que enemigo fue disparado de que fila y columna para desaparecerlo.

Luego se detecta el disparo de los enemigos hacia la nave, aquí se dan tres condiciones porque al ser disparada la nave por tres ocasiones el juego termina, caso contrario el juego sigue, en cada colisión la nave desaparece un instante y luego reaparece.

Si los enemigos aún no han sido eliminados, pero llegan a la posición de la nave el juego termina.

```
def check_collisions(self):
    # todo detecta la colision entre laser
    colicion_entre_laser = sprite.groupcollide(self.bullets, self.enemyBullets, True, False)
    if colicion_entre_laser:
        for value in colicion_entre_laser.values():
            for currentSprite in value:
                self.enemyBullets.remove(currentSprite)
                self.allSprites.remove(currentSprite)

            # todo detecta si nuestra nave le llego al invasor con el laser
    enemigo_destruido_laser = sprite.groupcollide(self.bullets, self.enemies, True, False)
    if enemigo_destruido_laser:
        for value in enemigo_destruido_laser.values():
            for currentSprite in value:
                self.sounds["invaderkilled"].play()
                self.killedRow = currentSprite.row #todo detecta cual invasor fue matado(fila)
                self.killedColumn = currentSprite.column#todo detecta cual invasor fue matado(columna)
                score = self.calculate_score(currentSprite.row)
                explosion = Explosion(currentSprite.rect.x, currentSprite.rect.y, currentSprite.row, False)
                self.explosionsGroup.add(explosion)
                self.allSprites.remove(currentSprite)
                self.enemies.remove(currentSprite)
                self.gameTimer = time.get_ticks()
                break
```

Ilustración 23 Implementación Clase SpaceInvaders Función 12

```
        # todo detecta si el invasor nos disparo a la nave
    nave_destruida_laser = sprite.groupcollide(self.enemyBullets, self.playerGroup, True, False)
    if nave_destruida_laser:
        for value in nave_destruida_laser.values():
            for playerShip in value:
                if self.lives == 3:
                    self.lives -= 1
                    self.livesGroup.remove(self.life3)
                    self.allSprites.remove(self.life3)
                elif self.lives == 2:
                    self.lives -= 1
                    self.livesGroup.remove(self.life2)
                    self.allSprites.remove(self.life2)
                elif self.lives == 1:
                    self.lives -= 1
                    self.livesGroup.remove(self.life1)
                    self.allSprites.remove(self.life1)
                elif self.lives == 0:
                    self.gameOver = True
                    self.startGame = False
                self.sounds["shipexplosion"].play()
                explosion = Explosion(playerShip.rect.x, playerShip.rect.y, 0, True)
                self.explosionsGroup.add(explosion)
                self.allSprites.remove(playerShip)
                self.playerGroup.remove(playerShip)
                self.makeNewShip = True
                self.shipTimer = time.get_ticks()
                self.shipAlive = False

      #todo  detecta si los invasores nos topa ala nave fin del juego
      if sprite.groupcollide(self.enemies, self.playerGroup, True, True):
          self.gameOver = True
          self.startGame = False

    todo fin del metodo de detectar la colision
```

Ilustración 24 Implementación Clase SpaceInvaders Función 12

La función crear una nueva nave sirve para cargar la imagen de la nave después de la colisión que tuvo con el disparo de un enemigo, esperando un pequeño tiempo.

```
def create_new_ship(self, createShip, currentTime):
    if createShip and (
            currentTime - self.shipTimer > 900):  #
        self.player = calseNaveEspacial()  # todo ca
        self.allSprites.add(self.player)
        self.playerGroup.add(self.player)
        self.makeNewShip = False
        self.shipAlive = True
```

Ilustración 25 Implementación Clase SpaceInvaders Función 13

La función fin de juego tiene implementado la carga de datos con el tiempo de juego y los puntajes que hizo el jugador en una nueva ventana, está establecido la posición de impresión y el tamaño de letra. Luego de un tiempo la ventana se cierra y se redirecciona a la venta de menú principal.

```
def create_game_over(self, currentTime):
    self.screen.blit(self.background, (0, 0))
    if currentTime - self.timer < 2999:  #todo para el mensaje de game over
        print(self.timer)
        self.gameOverText.draw(self.screen)

        self.titulo_puntaje = Text(FONT, 40, "PUNTAJE:", BLUE, 200, 500)  # todo esc
        self.titulo_puntaje.draw(self.screen) #todo dibuja puntaje en la pantalla el
        self.texto_puntaje = Text(FONT, 50, str(self.score), GREEN, 500, 500)  # tod
        self.texto_puntaje.draw(self.screen)#todo escribe en la pantalla el puntaje

        self.titulo_tiempoT = Text(FONT, 40, "TIEMPO:", BLUE, 200, 400)  # X,Y
        self.titulo_tiempoT.draw(self.screen)
        self.texto_tiempoT = Text(FONT, 50, str(self.timer/1000), GREEN, 500, 400)
        self.texto_tiempoT.draw(self.screen)

    if currentTime - self.timer > 3000:  # todo para que se reinicie el main princip
        self.mainScreen = True

    for e in event.get():
        if e.type == QUIT:
            sys.exit()
```

Ilustración 26 Implementación Clase SpaceInvaders Función 14

La función main tiene la carga de todos los textos que corresponde a la pantalla menú, los que corresponde a la pantalla juego, la imagen de fondo de las pantallas y también tiene el cronometro del juego, además, tiene el reseteo del juego para que los enemigos vuelvan a estar en posición para un nuevo juego.

Hay que tomar en cuenta que en todas las funciones ya mencionadas se hace el llamado de las primeras funciones, hay que notar la relación que tenían entre ellas.

```
def main(self):
    while True:
        if self.mainScreen:
            self.reset(0, 3, True)
            self.screen.blit(self.background, (0, 0))
            # todo iniciar todos los titulos del menu principal
            self.titulo_programacion.draw(self.screen)
            self.titulo_SpaceInvader.draw(self.screen)
            self.titulo_pres_tecla.draw(self.screen)
            self.titulo_autores1.draw(self.screen)
            self.titulo_autores2.draw(self.screen)
            self.titulo_autores3.draw(self.screen)
            self.titulo_boton_inicio.draw(self.screen)
            self.titulo_boton_salir.draw(self.screen)

            self.create_main_menu()

        elif self.startGame:
            if len(self.enemies) == 0:
                currentTime = time.get_ticks()
                if currentTime - self.gameTimer < 3000:
                    self.screen.blit(self.background, (0, 0))
                    self.scoreText2 = Text(FONT, 20, str(self.score), GREEN, 85, 5)
                    self.scoreText.draw(self.screen)
                    self.scoreText2.draw(self.screen)
                    self.nextRoundText.draw(self.screen)
                    self.check_input()
```

Ilustración 27 Implementación Clase SpaceInvaders Función 15

```
        if currentTime - self.gameTimer > 3000:
            # Move enemies closer to bottom
            self.enemyPositionStart += 35
            self.reset(self.score, self.lives)  # todo resetea el puntaje
            self.make_enemies()
            self.gameTimer += 3000
        else:
            #todo imprime en la pantalla de juego
            currentTime = time.get_ticks()
            self.play_main_music(currentTime)
            self.screen.blit(self.background, (0, 0))
            #todo cronometro
            self.Tiempo = time.get_ticks() / 1000
            self.texto_tiempoT = Text(FONT, 20,"TIME:  "+ str(self.Tiempo), YELLOW, 350, 20)
            self.texto_tiempoT.draw(self.screen)
            #todo fin del cronometro
            self.scoreText2 = Text(FONT, 20, str(self.score), GREEN, 85, 5)
            self.scoreText.draw(self.screen)
            self.scoreText2.draw(self.screen)
            self.livesText.draw(self.screen)
            self.check_input()
            self.allSprites.update(self.keys, currentTime)
            self.explosionsGroup.update(self.keys, currentTime)
            self.check_collisions()
            self.create_new_ship(self.makeNewShip, currentTime)


            if len(self.enemies) > 0:
                self.make_enemies_shoot()


            if len(self.enemies) > 0:
                self.make_enemies_shoot()

    elif self.gameOver:
        currentTime = time.get_ticks()
        # todo resetiar al enemigo empiece en la posicion
        self.enemyPositionStart = self.enemyPositionDefault
        self.create_game_over(currentTime)


    display.update()
    self.clock.tick(60)
```

Ilustración 28 Implementación Clase SpaceInvaders Función 15

Por último, se hace el llamado a las funciones principales.

```
if __name__ == '__main__':
    game = SpaceInvaders()
    game.main()
```

Ilustración 29 Implementación Clase SpaceInvaders Llamado a función

5.2. CREACIÓN ARCHIVO EJECUTABLE

Para realizar un archivo ejecutable de este programa realizamos lo siguiente: creamos un nuevo archivo .py dentro de la misma carpeta en la cual se encuentra el programa que hemos implementado.

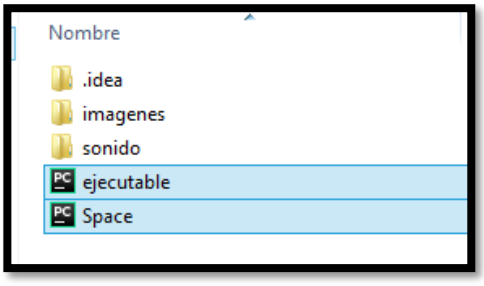

Ilustración 30 Archivos .py en carpeta general del Juego

Este nuevo archivo tendrá el siguiente código:

```
import sys
import os
from cx_Freeze import setup, Executable

build_exe_options = {'packages': []}
base = "Win32GUI"

exe = Executable(
      script="cc.py",
      base="Win32GUI",


)
setup(
      name = "Space Invaders",
      options = {'build_exe': build_exe_options},
      executables = [Executable("Space.py", base=base)])
```

Ilustración 31 Código para crear archivo ejecutable

El mismo que permitirá convertir un archivo .py en un archivo .exe.

Hay que abrir la ventana de comando del computador y ejecutar la siguiente línea de código **python ejecutable.py build** *tomando en cuenta que debe estar en el directorio en el cual se encuentra este archivo.*

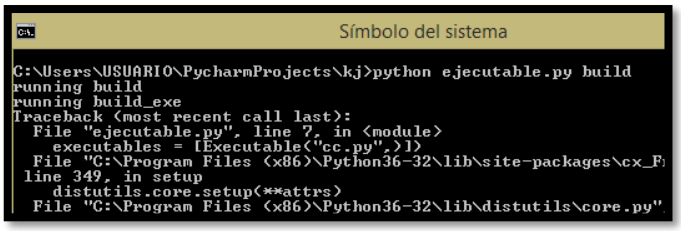

Ilustración 32 Creación archivo ejecutable con la línea de comandos

En caso de todo estar bien se crea una nueva carpeta llamada build y dentro de esta se encuentra el archivo ejecutable de acuerdo con el nombre que se le hay dado.

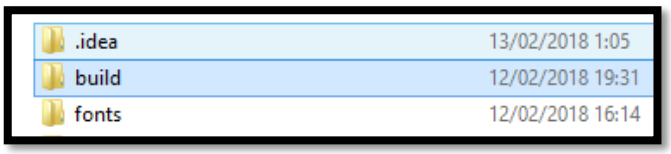

Ilustración 33 Carpeta build contiene archivo ejecutable

Ilustración 34 Archivo Ejecutable

6. PROGRAMAS UTILIZADOS

Para el desarrollo del proyecto los programas que hemos utilizado son:

Tabla 1 Programas Utilizados en el proyecto

PROGRAMAS	VERSIÓN
Python	*3.4.3*
Pygame	*1.9.3*
Pycharm	*2017.2*
cx_Freeze	*4.3.4*

7. MANUAL DE INSTALACIÓN

7.1 INSTALACIÓN DE PYTHON

Para instalar Python, primero hay que descargarlo; luego continuar con los siguientes pasos.

1. *Ejecutar Python, y observar que se despliega una venta en la cual se selecciona la opción* **Install for all users.** *Y presionar* **Next.**

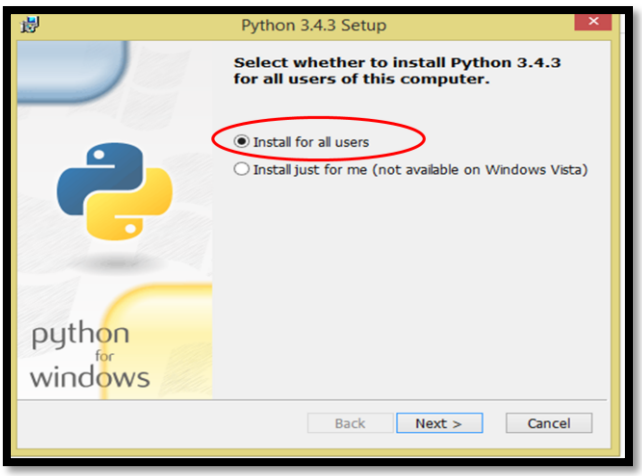

Ilustración 35 Seleccionar instalar Python

2. *Seleccionar el directorio destino y presionar* **Next.**

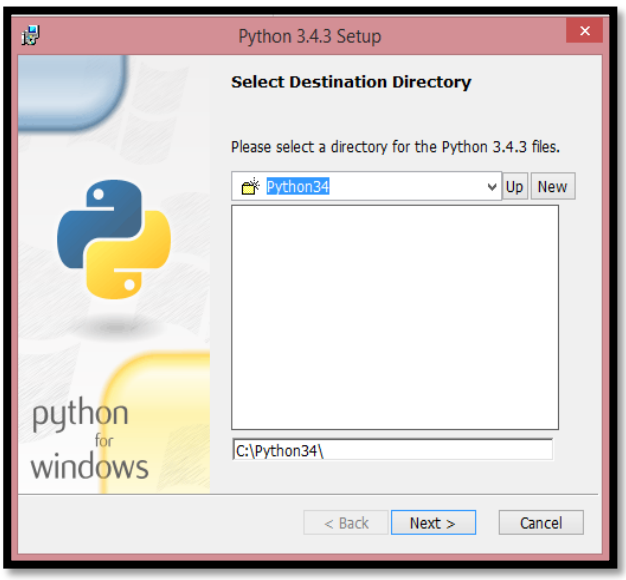

Ilustración 36 Seleccionar directorio destino

3. *En la siguiente ventana se puede observar las librerías e intérprete de Python. Seleccionar Python y dar clic en* **Next.**

Ilustración 37 Interprete de Python y librerías

4. *Esperar unos minutos mientras se instala Python.*

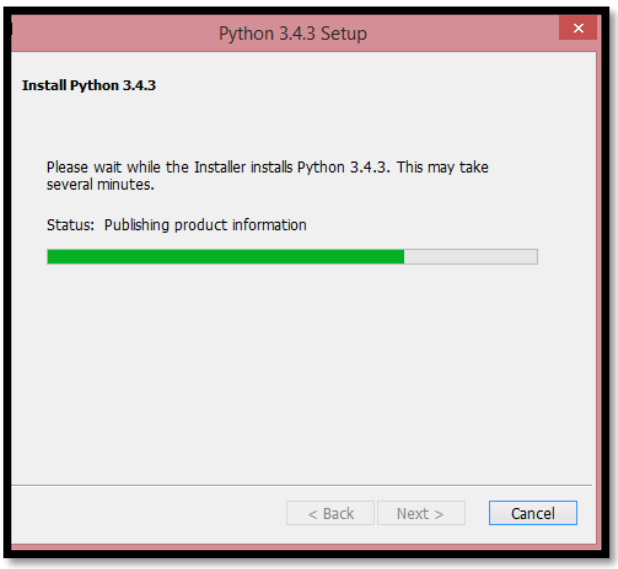

Ilustración 38 Instalación de Python

5. *Una vez que se termina la instalación dar clic en* **Finish.**

Ilustración 39 Fin de la instalación de Python

7.2 INSTALACIÓN DE PYCHARM

Para instalar Pycharm primero hay que descargarlo y sobre todo hay que crear una cuenta en **jetbrains** *para tener una licencia para su uso.*

1. *Al ejecutar Pycharm se observa una ventana de inicio de instalación, hacer clic en* **Next.**

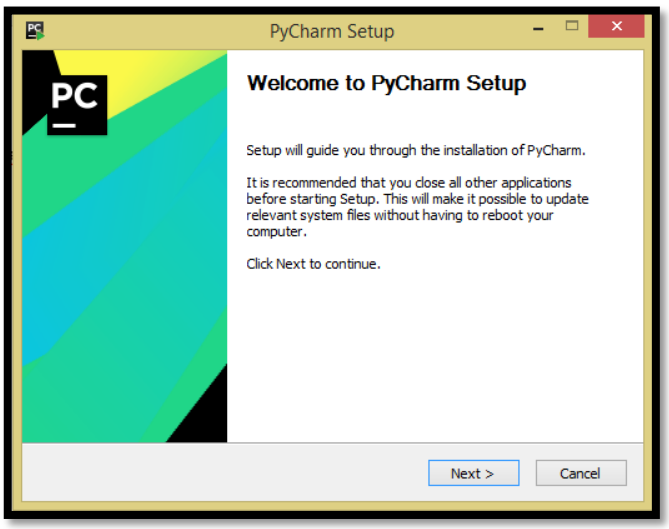

Ilustración 40 Inicio de instalación de Pycharm

2. *Elegir la ubicación de la instalación, y dar clic en* **Next.**

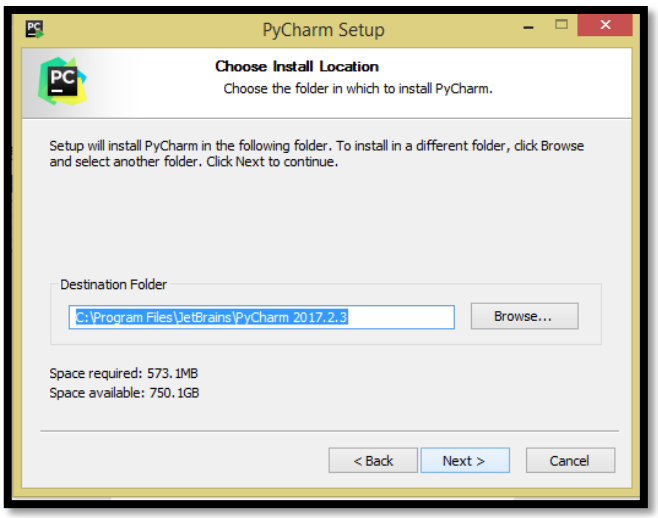

Ilustración 41 Elegir ubicación de instalación

3. *Seleccionar el programa de acuerdo con los requerimientos del ordenar que se disponga. Y hacer clic en* **Next.**

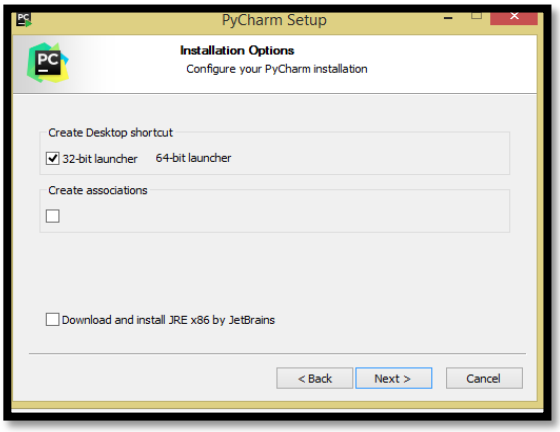

Ilustración 42 Opciones de Instalación

4. *Elegir del menú la carpeta de inicio, para la instalación. Y dar clic en* **Install.**

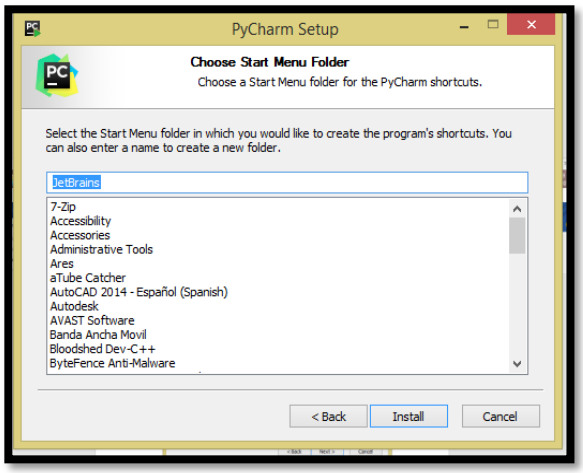

Ilustración 43 Elegir carpeta de inicio

5. *Esperar unos minutos hasta que finalice la instalación.*

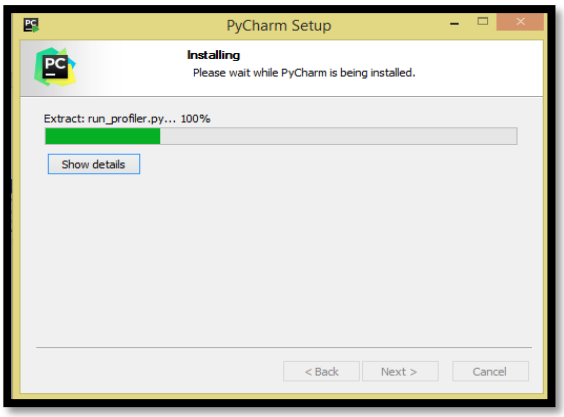

Ilustración 44 Esperar que finalice la instalación

6. *Al finalizar la instalación dar clic en* **Finish.**

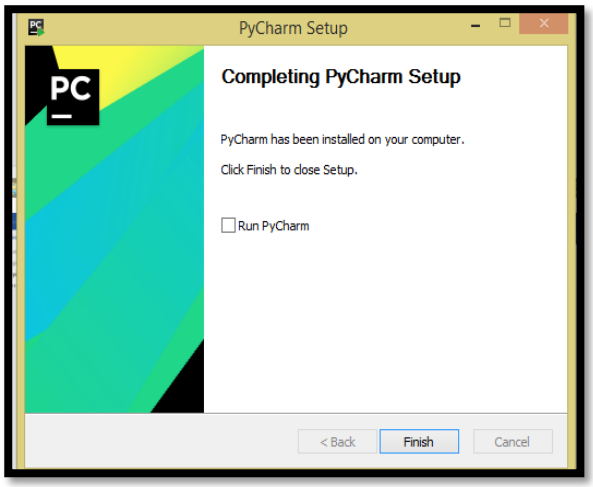

Ilustración 45 Fin de la instalación

7. *Ejecutar* **PyCharm** *para conocer su funcionamiento.*

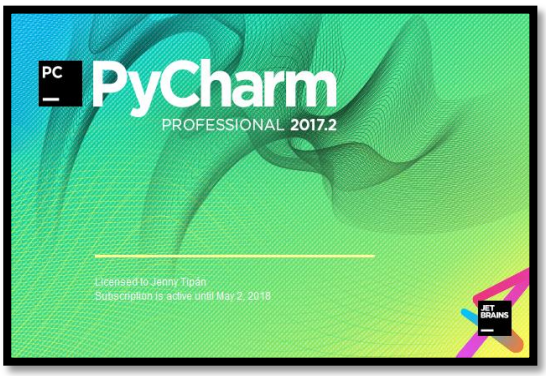

Ilustración 46 Ejecución de PyCharm

8. *Se puede observar una nueva ventana en esta seleccionar la opción que desee; ya sea para crear o abrir un proyecto.*

Ilustración 47 Ventana de Opciones

9. *Elegir el tipo de proyecto que se desea realizar y dar clic en* **Continue.**

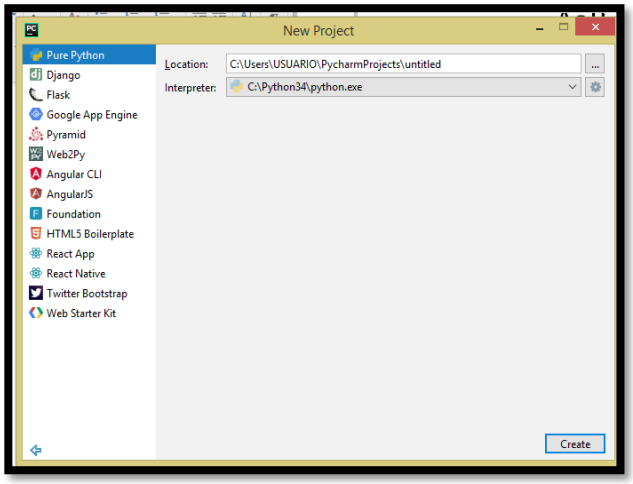

Ilustración 48 Selección de proyecto

10. *En File se encuentra la opción crear un nuevo proyecto, se le da un nombre y se empieza con la implementación de programas.*

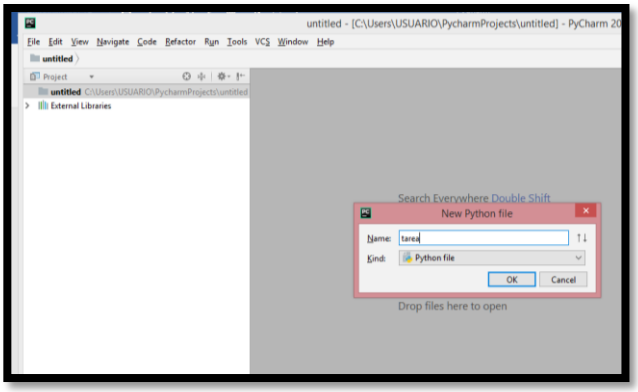

Ilustración 49 Creación de un proyecto en PyCharm

7.3 INSTALACIÓN DE SOURCETREE

Para la instalación de esta aplicación, primero hay que descargarla y luego seguir con los siguientes pasos:

1. *Primero hay que seleccionar* **I Agree** *que indica que se acepta los términos de licencia, y dar clic en* **Continue.**

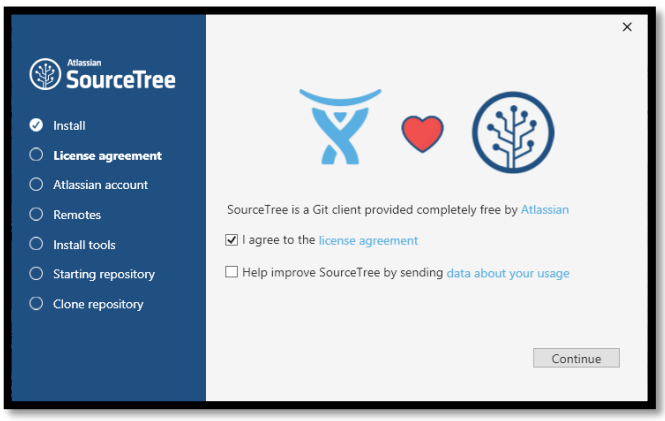

Ilustración 50 Aceptar términos de licencia de SourceTree

2. *Seleccionar la opción* **Use an existing account**

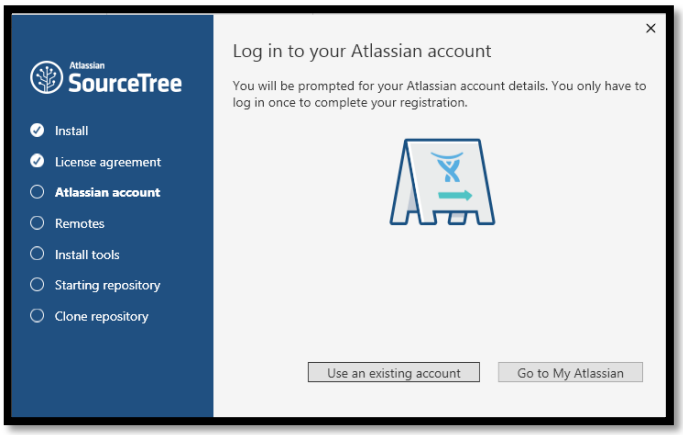

Ilustración 51 Selección de opción Use an existing account

3. *Seleccionar la opción* **Login in with Google**

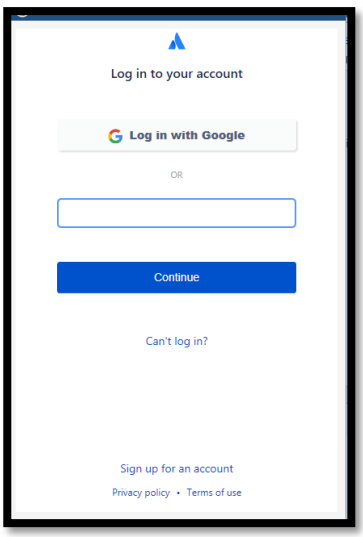

Ilustración 52 Seleccionar Login in with Google

4. *Ingresar la dirección de la cuenta de Gmail personal.*

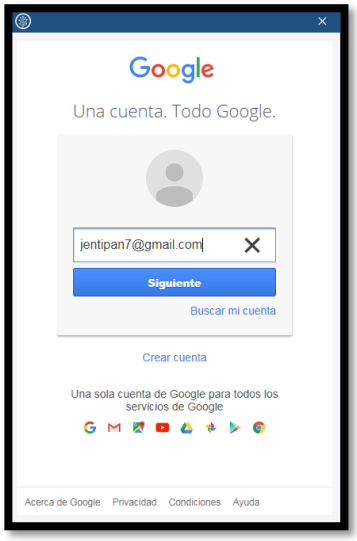

Ilustración 53 Ingresar dirección de correo electrónico

5. *Ingresar contraseña de la cuenta de Gmail*

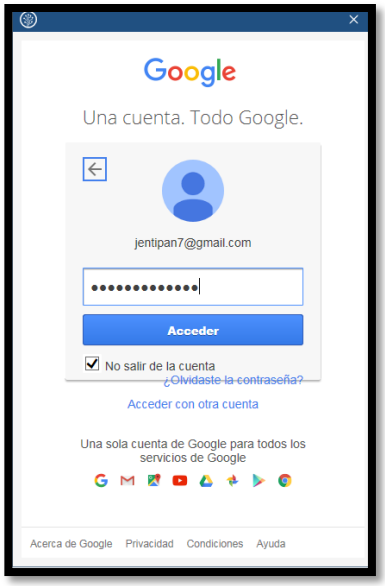

Ilustración 54 Ingresar contraseña de la cuenta

6. *Una vez verificada la cuenta se observa una nueva venta, en esta dar clic en* **Continue.**

Ilustración 55 Verificación de cuenta de Gmail

7. *De la siguiente venta hay que elegir la opción* **GiHub**. *Dar clic en* **Continue.**

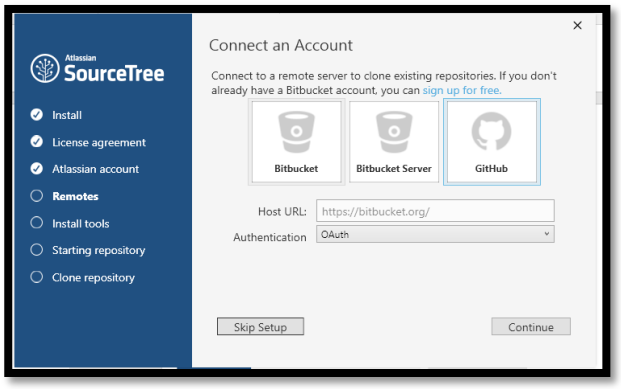

Ilustración 56 Seleccionar Cuenta

8. *Si se tiene una cuenta de* **GiHub** *ingresar los datos de la cuenta, y en caso de no tenerla crear una. Hay que ingresar el correo electrónico y su contraseña.*

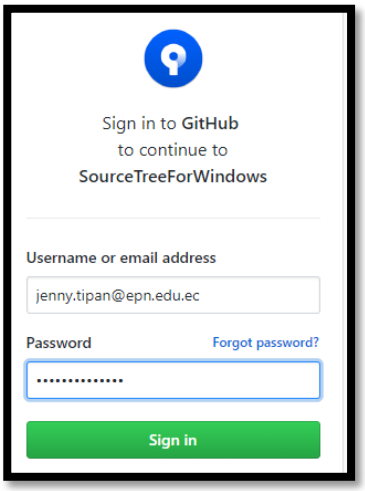

Ilustración 57 Ingresar datos de cuenta GiHub

9. *Una vez verificada la cuenta, empieza la descarga de herramientas. Y se finaliza la instalación.*

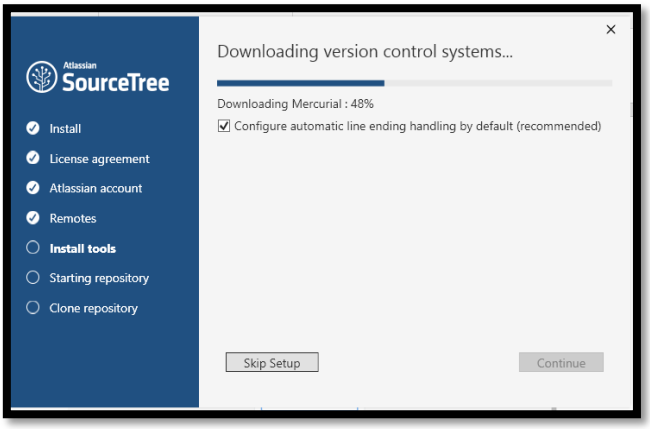

Ilustración 58 Instalación de herramientas de SourceTree

7.4 INSTALACIÓN DE CX_FREEZE

1. *Descargar el programa cx_Freeze de acuerdo con la versión de Python que dispongan.*

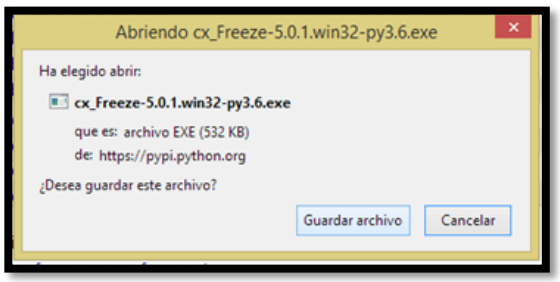

Ilustración 59 Descarga programa cx_Freeze

2. *Al ejecutar el programa empieza la instalación, dar clic en siguiente.*

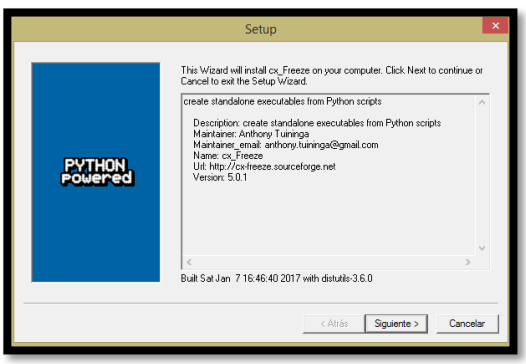

Ilustración 60 Dar clic en siguiente

3. *En la nueva ventana observamos en donde se va a realizar la instalación, dar clic en siguiente.*

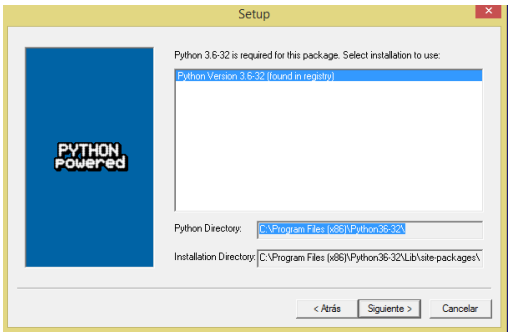

Ilustración 61 Observar en donde se descargará el programa

4. *En la nueva ventana hay un mensaje de instalación, dar clic en siguiente.*

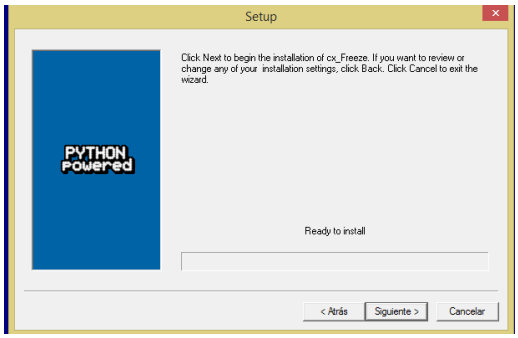

Ilustración 62 Seleccionar siguiente y esperar que termine la descarga

5. *Esperamos unos segundos mientras termina la instalación. En la nueva venta que aparece dar clic en Finalizar.*

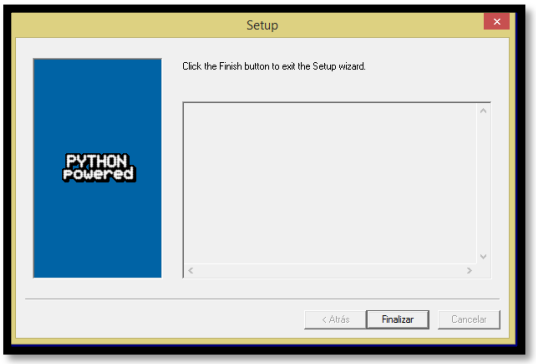

Ilustración 63 Dar clic en finalizar

8. REGISTRO DE ACTIVIDADES

Tabla 2 Tabla de Registro de actividades

MIEBROS DEL GRUPO	ACTIVIDADES		
ANDRÉS CORO	*Implementación carga de enemigos*	*ELABORACIÓN DEL INFORME*	*PRESENTACIÓN EN POWER POINT*
	Implementación reloj		
	Implementación de puntajes		
	Implementación de reseteo de juego		
JORGE IZA	*Implementación carga de imágenes*		
	Implementación de Sonidos		
	Implementación de Colisiones		
	Unión general del código		
JENNY TIPÁN	*Descarga de imágenes y audios*		
	Implementación de Botones		
	Implementación creación de textos		
	Implementación pantalla de menú		

9. CRONOGRAMA DE ACTIVIDADES

Tabla 3 Cronograma de Actividades

CRONOGRAMA DE ACTIVIDADES												
N.	FECHA / ACTIVIDADES	ENERO					FEBRERO					
		18	22	25	29	31	1	5	7	8	12	15
1.	*Propuesta de Proyecto*											
2.	*Descarga de programas*											
3.	*Descarga de imágenes y audios*											
4.	*Diseño de la Interfaz (Pantalla)*											
5.	*Implementar carga de imágenes (fondo de pantalla y nave)*											
6.	*Movimiento de la nave (Izquierda - Derecha)*											
7.	*Efectos de disparo*											
8.	*Implementación de cronómetro y sonido*											
10.	*Presentación Avance Proyecto*											
11.	*Implementación de enemigos*											
12.	*Detección de Colisiones*											
13.	*Implementación pantalla de Menú y Puntajes*											
14.	*Implementación total código*											
15.	*Realización del informe y archivo ejecutable*											
16.	*Presentación Final proyecto*											

10. CONCLUSIONES

Luego de realizar la implementación de este programa hemos sacado las siguientes conclusiones:

- *Es de vital importancia tomar en cuenta las versiones de los programas antes de iniciar la implementación, por es muy complicado tratar de solucionar problemas de este tipo una vez que se tiene el código terminado.*
- *Para realizar este programa utilizamos la librería pygame que al ser un módulo que sirve para creación de videojuegos nos permitió darle visualmente a nuestro programa un diseño muy agradable.*
- *Trabajar con una cuenta de GitHub nos proporciona ayuda para compartir las actualizaciones que realizamos de nuestro código.*

11. PROBLEMAS ENCONTRADOS

En el desarrollo del código el problema que encontramos fue la implementación del cronometro, ya que no logramos hacer que reinicie el reloj en caso de salir del juego y nuevamente optar por jugar.

Ilustración 64 Puntaje fin de juego

Ilustración 65 Tiempo sin reiniciar en nuevo juego

El segundo problema que encontramos fue la realización de un archivo ejecutable ya que la versión de Python que usamos al parecer no era compatible con el programa cx_Freeze y no nos permitió realizar este archivo.

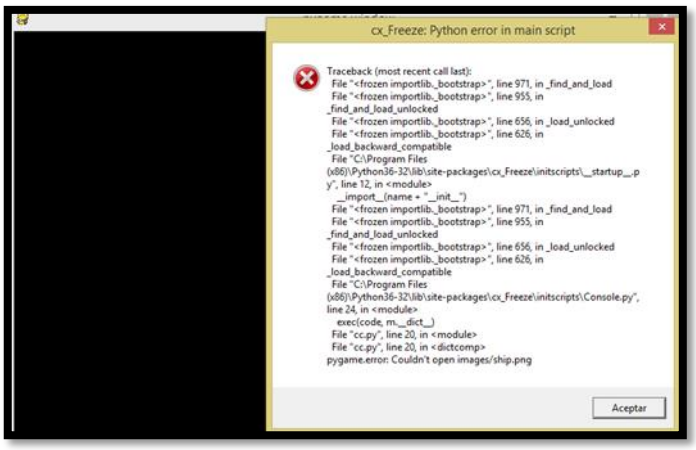

Ilustración 66 Problema ejecución de archivo .exe

12. RECOMENDACIONES

- *Verificar la compatibilidad entre las versiones de los programas que se van a utilizar, ya que al final se pueden presentar problemas que son un tanto complicados de resolver, como cambiar de versión de Python lo que indica que se tiene que cambiar la versión de todos los programas utilizados.*
- *Usar una cuenta de GitHub para compartir archivos, ya que en este se puede ir conservando versiones anteriores de los mismos, que serían útiles en caso de que se quiera trabajar con una versión anterior.*

13. RESULTADOS

Ya implementado el código podemos observar que el programa funciona correctamente.

Primero se abre la ventana de inicio o menú dentro de la cual están dos botones para iniciar juego y salir.

Ilustración 67 Ventana de menú

Al presionar el botón inicio podemos ver la pantalla de juego, en donde están los enemigos, la nave, el tiempo, las vidas y el puntaje.

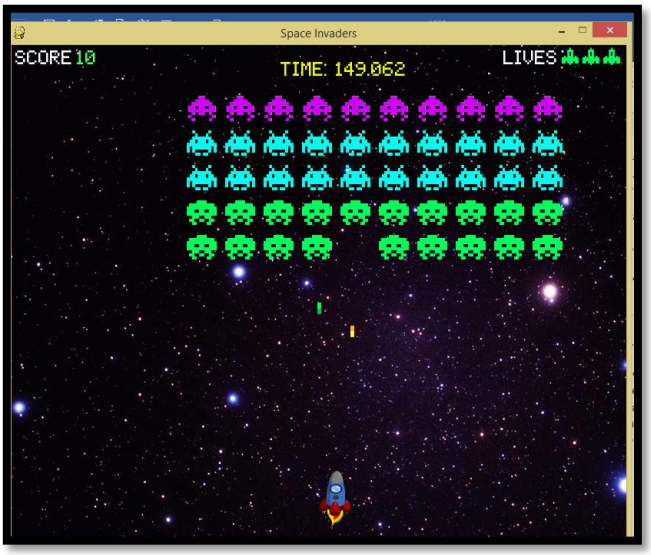

Ilustración 68 Pantalla de Juego

Por último, tenemos la ventana de puntaje en la cual se observa el tiempo de juego y el puntaje alcanzado.

Ilustración 69 Ventana de Puntaje

14. REFERENCIAS

[1] *«Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Pygame. [Último acceso: 01 Enero 2018].*

[2] *«Python Mania,» [En línea]. Available: https://www.pythonmania.net/es/2010/03/23/tutorial-pygame-introduccion/. [Último acceso: 01 Enero 2018].*

[3] *[En línea]. Available: https://es.wikipedia.org/wiki/Pygame. [Último acceso: 12 Febrero 2018].*

[4] *[En línea]. Available: https://translate.google.com.ec/translate?hl=es&sl=en&u=https://readthedocs.org/projects/cx-freeze/downloads/pdf/latest/&prev=search. [Último acceso: 12 Febrero 2018].*

[5] *[En línea]. Available: https://www.genbetadev.com/sistemas-de-control-de-versiones/sourcetree-cliente-gui-para-manejar-repositorios-git-o-mercurial-llegara-en-breve-a-windows. [. [Último acceso: 12 Febrero 2018].*