UNIVERSIDAD LUTERANA SALVADOREÑA
FACULTAD DE CIENCIAS DEL HOMBRE Y LA NATURALEZA
LICENCIATURA EN CIENCIAS DE LA COMPUTACIÓN

TEMA:

Space Invaders

INTEGRANTES:

Bryan Alexander Vázquez Hernández

Carlos Mauricio Guerrero Granada

ASIGNATURA:

Ingeniería del Software

DOCENTE:

Lesbia María Mancía de Maradiaga

CICLO/AÑO:

II-2020

FECHA:

06-12-20

Introducción

En el siguiente informe se le presentará un proyecto con el Lenguaje de programación Python, se abordan los principales aspectos en la elaboración del proyecto, se tomarán puntos como el proceso que se llevó a cabo para la estructura de este programa que recursos fueron utilizados, sus limitaciones y su desarrollo que ha tenido conforme el paso del tiempo.

Se observa una información con respecto al juego, un poco de cómo dio origen a la idea a realizar, el propósito el por qué se escogió este proyecto a realizar con qué fines se va a realizar. De la misma manera se intenta cubrir información sobre los requerimientos del software, las tecnologías que serán implementadas en la aplicación y el desarrollo ágil, es decir, la forma o modelo en el que se desarrollará el mismo.

Contenido

Objetivos.

General:

Abarcar información importante sobre el software en desarrollo (Naves Espaciales).

Específicos:

- Definir la manera en la que se desarrollará el software.
- Analizar los requerimientos del sistema.
- Especificar las tecnologías que se implementan en el sistema.

Justificación.

Se elabora este proyecto primeramente con fines educativos y prácticos, para que se pueda conocer de la estructura o los requerimientos necesarios que se tiene que tomar en cuenta al momento del desarrollo de un sistema, se hará con el lenguaje de Python para más facilidad y precisión.

Empresa: Game Store

Space Invaders nació en 1978 por obra de Toshihiro Nishikado, un diseñador japonés de la Taito Corporation. es una sociedad de una comunidad de programadores, la empresa se dedica a la elaboración de proyectos tales como: aplicaciones, juegos, programas, sitios web, paginas publicitarias, entre otros.

Comenzando su progreso de apertura en un pequeño local con poco personal el nombre del fundador es Isaac Villa un licenciado en la programación, conforme fueron pasando los años su empresa poco a poco fue creciendo en el desarrollo de diferentes juegos.

Hoy en día la empresa es una sociedad muy conocida llevando ya 22 años de ser fundada, con una galería de diversos programas, aplicaciones, sitios web, entre otros.

Entre ellos encontramos Space Invaders este es un juego basados en naves espaciales que su único propósito es matar alienígenas que viene descendiendo este juego era de maquinillas con un algoritmo muy sencillo hoy en día se puede ver en diferentes móviles y para máquinas de escritorio. Con una interfaz gráfica bastante mejorada gracias a la tecnología de hoy en día.

Visión

Una empresa desempeñada en realizar los mejores programas para un ordenador, aplicaciones, juegos y más para el móvil tanto para viejas como nuevas generaciones.

Misión

Brindar los mejores servicios, actualizaciones, mantenimiento y otros en los diferentes programas, aplicaciones, sitios web creadas o patentadas por nuestra sociedad.

Descripción del proyecto.

Nuestro proyecto es un programa con función videojuego con una interfaz sencilla es 2D, es un juego de naves espaciales como en las maquinillas de arcade que se encontraban en algunos centros comerciales donde los niños se iban jugar junto a sus amigos.

Nuestro juego está basado en los juegos de años pasados su nombre “Space Invaders” un juego de naves espaciales con invasores que descienden desde arriba y el jugador era representado como una nave por el espacio.

De ahí el nombre de nuestro proyecto “Naves espaciales” que tiene una similitud al juego pero con diferentes algoritmos e interfaz.

Aplicaciones para el proyecto.

Python.

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python.

Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como Benevolente Dictador Vitalicio (en inglés: Benevolent Dictator for Life, BDFL); sin embargo el 12 de julio de 2018 declinó de dicha situación de honor sin dejar un sucesor o sucesora y con una declaración altisonante.

Sublime Text 2.

Sublime Text 2 es un editor de código al estilo TextMate, Kate o Redcar, su interfaz es limpia e intuitiva y soporta el uso de Snippets, Plugins y sistemas de construcción de código (Build Systems).

En un principio fue creado por Jon Skinner como una extensión rica en features de vim pero poco a poco fue adoptando identidad propia, debido a ello, Sublime Text 2 aún conserva un modo de edición tipo vi llamado “Vintage mode“.

El modo Vintage está desactivado por defecto pero puede ser activado cambiando las opciones del editor en la configuración global.

Sublime Text 2 está escrito en C++ y Python, incluye una implementación de CPython2.6 embebida, así como una consola que no es sino un intérprete de Python desde el cual podemos ejecutar comandos y realizar tareas de retrospección y hacks múltiples.

Requerimientos del Sistema

Requerimientos Funcionales

1. **Mostrar una pantalla de inicio (Naves espaciales) al ejecutar el programa.**

    1.1 La pantalla de inicio mostrará un menú el cual deberá contener opciones de: Iniciar Juego, Récord de puntajes y opción de salir del juego. Contenido extra (por si se llegase a alcanzar el primer contenido): “ajustes”, “opción de escoger personajes”.

    1.2 El menú tendrá una canción de fondo.

2. **Mostrar un tutorial.**

    2.1 El usuario al escoger la opción “Iniciar Juego” podrá ver un tutorial de como mover al personaje (Nave espacial) con el teclado.

3. **El usuario podrá iniciar el juego.**

    3.1 Las teclas de movimiento son: W, A, S, D y ESPACIO para disparar

    3.2 El juego inicia con el personaje (Nave) en el centro de la pantalla.

    3.3 La pantalla será un fondo negro con animaciones.

    3.4 Los enemigos (naves enemigas) aparecen en la parte de arriba y descienden mientras disparan su láser.

    3.5 El sistema mostrará la barra indicadora de vida del personaje (Nave) en la parte inferior del mismo con un color verde.

    3.6 El sistema mostrará el puntaje que vaya acumulando el jugador en la parte superior izquierda.

    3.7 El sistema muestra el incremento de niveles.

    3.8 El sistema reproduce música de fondo para ambientar.

    3.9 El sistema tiene efectos de sonido de disparos, explosiones y de perder la partida.

4. **El usuario perderá la partida su la vida llega a 0**

    4.1 Cuando la barra de vida sea completamente de color rojo el juego finaliza dando como mensaje “Game Over” para indicar que ha perdido la partida.

4.2 El juego reproduce un sonido acompañando el mensaje de “Game Over”.

5. **El usuario podrá observar el puntaje obtenido.**

    5.1 Se muestra una pantalla con los puntajes obtenidos tanto en la partida, como en las últimas jugadas.

6. **El usuario podrá salir del juego.**

Al momento que el usuario presione la tecla de “esc” podrá finalizar el juego o cerrar el programa.

Requerimientos No Funcionales

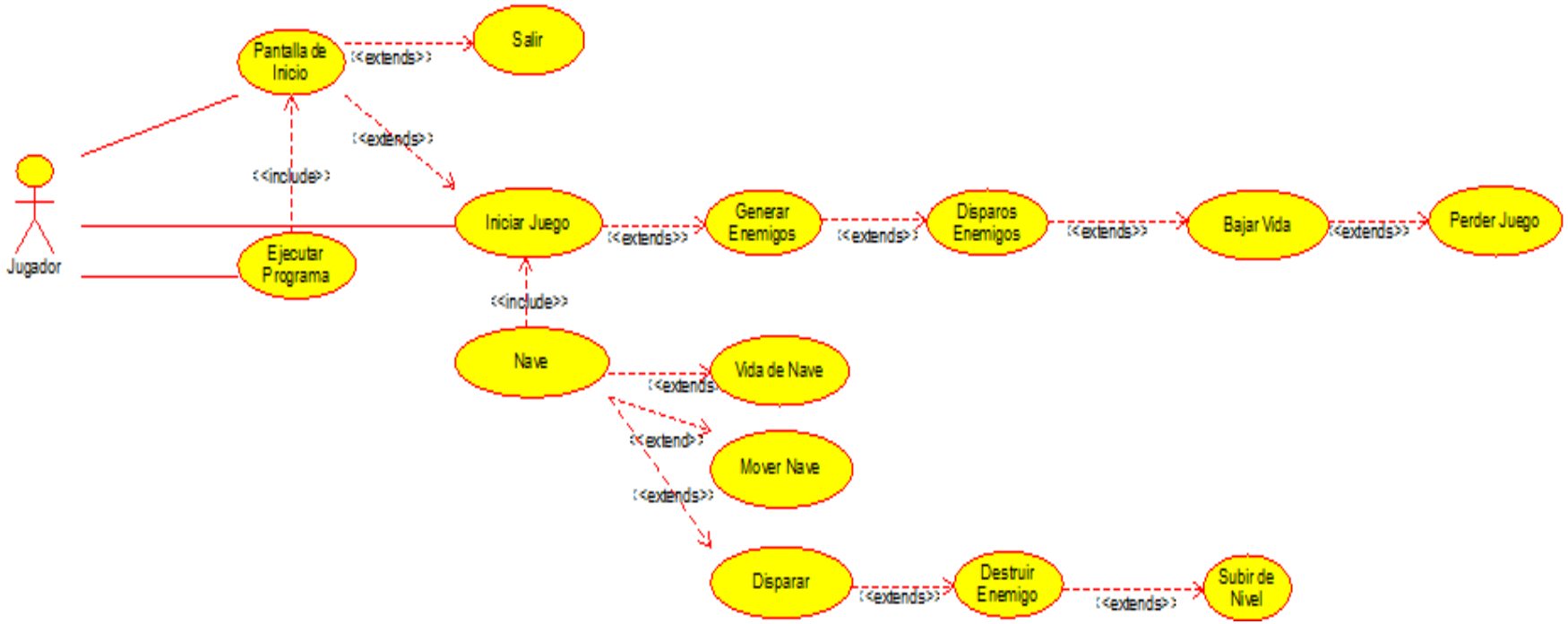
1. El programa no podrá sobrepasar la velocidad de ejecución que viene por defecto. (FPS)
2. No es compatible para dispositivos smartphone. Ni otra plataforma que no sea PC.
3. El sistema de volumen o sonido no puede ser cambiados por el usuario.
4. Por defecto el idioma de esta aplicación viene en español, no posee otros idiomas.
5. Los botones a utilizar ya vienen predeterminados, no es posible que el usuario pueda cambiarlos.
6. El aumento de vidas o barra de vida no excede ni cambia con el progreso de tiempo ejecutado en el programa.
7. El programa no conserva ni guarda datos obtenidos después de cerrar o dar por finalizado la ejecución del programa (puntaje o récords).

Desarrollo Ágil

DESCRIPCIÓN DE LA TAREA	RESPONSABLE	FECHA DE INICIO	FECHA FINAL	DIAS	ESTADO
Mostrar una pantalla de inicio					
Diseñar pantalla	Bryan Vázquez	22/08/20	24/08/20	2	Realizado
Mostrar en pantalla: Iniciar juego, Salir del Juego y ver Tutorial	Bryan Vázquez	26/08/20	30/08/20	4	Realizado
Reproducir música de fondo en el menú	Carlos Guerrero	09/10/20	09/10/20	1	Pendiente
Mostrar un tutorial					
Mostrar un tutorial que indique las teclas a utilizar para jugar	Carlos Guerrero	05/10/20	07/10/20	2	Pendiente
Iniciar Juego					
Hacer que el personaje se mueva con las teclas W, A, S, D	Bryan Vázquez	02/09/20	06/09/20	4	Realizado
Hacer que el personaje aparezca en el medio de la pantalla	Bryan Vázquez	07/09/20	10/09/20	3	Pendiente
Agregar un fondo negro en la pantalla	Bryan Vázquez	31/08/20	31/08/20	1	Realizado
Hacer que los enemigos aparezcan en la parte superior	Carlos Guerrero	11/09/20	14/09/20	3	Pendiente
Mostrar barra indicadora de vida	Carlos Guerrero	17/09/20	19/09/20	2	Pendiente
Mostrar cuando el usuario sube de nivel	Carlos Guerrero	23/09/20	24/09/20	1	Pendiente
Reproducir música de fondo en el juego	Carlos Guerrero	09/10/20	09/10/20	1	Pendiente
Reproducir efectos especiales	Carlos Guerrero	10/10/20	10/10/20	1	Pendiente
Perder la partida					
Mostrar el mensaje “Game Over” en la pantalla	Carlos Guerrero	28/09/20	28/09/20	1	Pendiente
Reproducir sonido al perder la partida	Carlos Guerrero	10/10/20	10/10/20	1	Pendiente
Salir del juego					
Definir tecla para poder finalizar el juego	Bryan Vázquez	12/10/20	13/10/20	1	Pendiente

DIAGRAMA DE CASOS DE USO.

Implementando Modelado de Caso de Uso en el proyecto Space Invaders



	Descripción de Caso de Uso Pantalla de Inicio	
Dependencias	• Opción Mostrar Récords de Usuario • Opción Iniciar Juego • Opción Salir de Juego	
Precondición	Haber ejecutado el programa correctamente	
Descripción	El sistema debe comportarse de acuerdo a la secuencia que se presentará a continuación.	
Secuencia Normal	1	Mostrar botones de la pantalla principal
	2	Esperar una orden del usuario
	3	Ejecutar la acción solicitada
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	Dependiendo de la opción seleccionada: • Récords: Ver puntajes recientes • Iniciar Juego: El usuario puede iniciar el juego • Salir del Juego: El usuario puede cerrar el programa si lo desea	

	Descripción de Caso de Uso Salir	
Dependencias	Salir del software	
Precondición	Haber ingresado al menú principal	
Descripción	El sistema permite al usuario salir del mismo	
Secuencia Normal	1	Clic botón salir (X)
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	El usuario puede cerrar el juego e interrumpir el proceso de ejecución del juego	

	Descripción de Caso de Uso Nave	
Dependencias	• Permite que aparezca la nave	
Precondición	Haber ejecutado el programa correctamente	
Descripción	El sistema debe comportarse de acuerdo a la secuencia que se presentará a continuación.	
Secuencia Normal	1	Mostrar botones de la pantalla principal
	2	Esperar una orden del usuario
	3	Ejecutar la acción solicitada
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	Dependiendo de la opción seleccionada: • Récords: Ver puntajes recientes • Iniciar Juego: El usuario puede iniciar el juego • Salir del Juego: El usuario puede cerrar el programa si lo desea	

	Descripción de Caso de Uso Iniciar Juego	
Dependencias	Cargar recursos del juego, naves, enemigos, música	
Precondición	Haber seleccionado la opción “Iniciar Juego”	
Descripción	El sistema debe comenzar el juego, la nave del jugador aparece en el centro de la pantalla, y los enemigos descienden de forma aleatoria a su vez, carga música de fondo y efectos especiales.	
Secuencia Normal	1	Nave aparece en el centro
	2	Suena música de fondo
	3	Aparecen enemigos
	4	Los enemigos disparan
	5	Efectos de sonido de disparos
	6	Puntaje y actualización de nivel
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	El usuario puede cerrar el juego e interrumpir el proceso de ejecución del juego	

	Descripción de Caso de Uso Vida de Nave	
Dependencias	• Mostrar la vida de la nave	
Precondición	Haber iniciado a jugar	
Descripción	El sistema agrega una barra de color verde debajo de la nave indicando la vida que esta tiene	
Secuencia Normal	1	Iniciar juego
	2	Aparece barra de vida
	3	Disminuye si recibe daño
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	• Permitir ver cuanta vida le queda a la nave	

	Descripción de Caso de Uso Mover Nave	
Dependencias	Designar flechas para mover la nave	
Precondición	Haber iniciado a jugar	
Descripción	El sistema permite a la nave moverse mediante las flechas del teclado	
Secuencia Normal	1	Apretar teclas, arriba, abajo, derecha e izquierda
	2	La nave se mueve donde se le indica
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	Evitar naves enemigas	

	Descripción de Caso de Uso Disparar	
Dependencias	• Disparar láser	
Precondición	Haber iniciado a jugar	
Descripción	Una de las características de la nave es disparar	
Secuencia Normal	1	Presionar tecla espacio
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	La nave destruye naves enemigas	

	Descripción de Caso de Uso Destruir Enemigo	
Dependencias	Destruir enemigos	
Precondición	Haber disparado o chocado con una nave enemiga	
Descripción	La nave puede disparar a las naves enemigas, o puede pasar sobre ellas y destruirlas también	
Secuencia Normal	1	Disparar con tecla espacio
	2	Moverse sobre las naves enemigas con las flechas direccionales
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	Subir de nivel	

	Descripción de Caso de Uso Subir de Nivel	
Dependencias	• Aumentar dificultad	
Precondición	Haber destruido naves enemigas	
Descripción	Destruir naves enemigas permite subir de nivel y a su vez aumentar la dificultad	
Secuencia Normal	1	Destruir enemigos
	2	Subir de nivel
	3	Aumentar la velocidad de reaparición de naves enemigas
Secuencia Alternativa	No destruir naves hace que el subir de nivel tarde	
Post Condición	Aumentar dificultad	

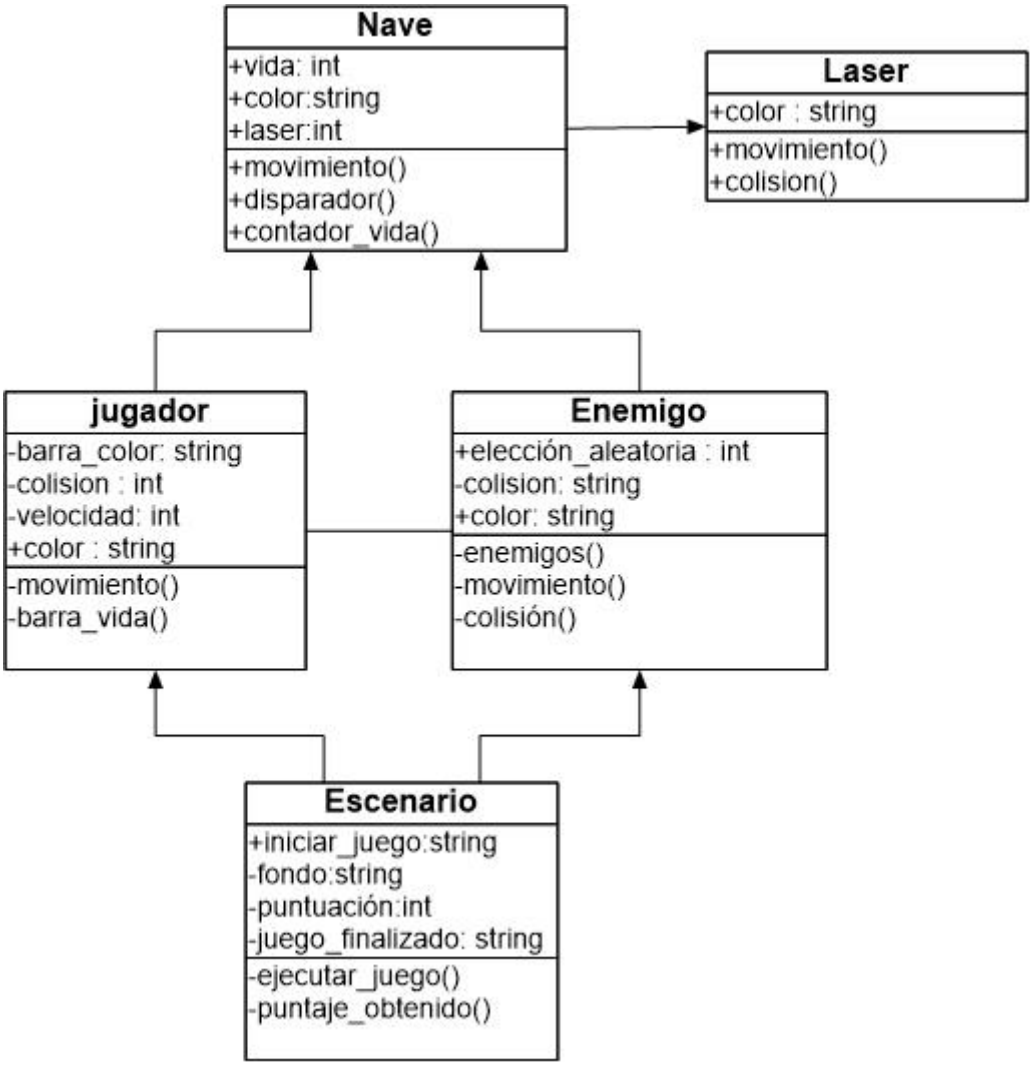
	Descripción de Caso de Uso Generar Enemigos	
Dependencias	Generar enemigos a los que destruir	
Precondición	Haber iniciado a jugar	
Descripción	El juego genera enemigos que deben destruirse para subir de nivel	
Secuencia Normal	1	Iniciar juego
	2	Enemigos automáticamente aparecen desde arriba
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	El usuario puede subir de nivel	

	Descripción de Caso de Uso Disparos Enemigos	
Dependencias	• Característica de enemigos para hacer difícil el juego	
Precondición	Haber generado enemigos	
Descripción	Los enemigos pueden disparar en línea recta y hacer daño al jugador	
Secuencia Normal	1	Generar enemigos
	2	Disparar
	3	Si impacta al jugador disminuir 10 de vida
Secuencia Alternativa	No hay secuencia alternativa	
Post Condición	• Hacer que jugador deba esquivar disparos	

	Descripción de Caso de Uso Bajar Vida	
Dependencias	Dificultad del jugador	
Precondición	Debe haber enemigos en la pantalla	
Descripción	Los enemigos pueden bajar vida al jugador, impactando con sus láseres, si el jugador pasa sobre ellas también baja su vida	
Secuencia Normal	1	Recibir disparo
	2	Pasar sobre naves enemigas
Secuencia Alternativa	El jugador esquiva disparos y no pasa sobre naves enemigas	
Post Condición	Si la vida baja a 0 pierde el juego	

	Descripción de Caso de Uso Perder Juego	
Dependencias	• Perder el juego	
Precondición	Haber recibido mucho daño en la nave	
Descripción	El jugador puede perder el juego si su vida llega a 0	
Secuencia Normal	1	Recibir daño
	2	Bajar a 0 la vida
Secuencia Alternativa	No existe secuencia alternativa	
Post Condición	• Volver al menú principal • Cerrar el juego	

DIAGRAMA DE CLASE (NAVES ESPACIALES).



Requerimientos de Confiabilidad y Seguridad (Naves Espaciales)

- El sistema es compatible con todos los Sistemas Operativos para ordenador, siempre y cuando sus componentes estén correctamente instalados.

- Se requiere una versión de Python 3 en adelante

- Se debe instalar una versión del módulo de Pygame para poder iniciar el videojuego.

- El videojuego no consume demasiados recursos de hardware, por lo cual no se necesita un ordenador potente.

- Funciona sin necesidad de acceso a internet.

- El videojuego solo necesita teclado para que el usuario lo pueda controlar.

- El videojuego no almacena datos del usuario, más que sus puntajes obtenidos, los cuales son eliminados al interrumpir la ejecución del mismo.

Identificación de Riesgo

RIESGO	PROBABILIDAD	EFECTOS
Las probabilidades de un posible virus	Moderado	Catastrófico
Las probabilidades de posibles fallos o errores en el software	Moderado	Grave
La elaboración del programa en gráficos puede ser irrelevante para la nueva generación	Moderado	Tolerable
Los requerimientos puedan cambiar en un cierto punto.	Moderado	Tolerable
El personal se incapacite o renuncie por diversos motivos.	Alto	Grave
Robo de información	Baja	Grave
Insuficientes controles desde la dirección del proyecto	Baja	Tolerable
La falta de conocimiento de una metodología o algún paso necesario para la ejecución del proyecto	Moderado	Tolerable

RIESGO	ESTRATEGIAS
Posible virus en software	Tener un buen conocimiento o implementación de protección.
Fallos o errores de sistema	Una alerta de clientes y solución inmediata.
Los gráficos	Posibles cambios en el entorno gráfico.
Requerimientos inestables	Investigación y seguimiento de posibles cambios.
Personal incapacitado o indispuesto.	Asesoría a candidatos nuevos con informe del proyecto.
Robo de información	No permitir que el sistema almacene ni tenga relación directa con archivos del sistema donde sea utilizado.
Insuficiente control	Hacer una buena planificación y verificar cada parte del sistema que pudiese generar un problema
Falta de conocimiento de metodologías	Investigar acerca de metodologías que agilicen y permitan ordenar el proceso de desarrollo del software

Alcances

Conocimiento básico de lo que es programación con Python con una serie de imágenes en movimiento y acerca de los modulo que vienen integrados con el mismo.

Limitaciones

Dificultades en la falta de conocimiento o entendimiento de algunos algoritmos de programación de Python y sus módulos.

Conclusiones

Un proyecto requiere de tiempo y organización, y hay muchas formas de poder hacerlo, las historias de usuario son una muestra clara de lo que es programarse, llevar tareas en orden, y dividir bien el trabajo.

Los requerimientos del sistema, nos da la pauta para el desarrollo de un sistema, debido a que como nos lo indica son los requerimientos, que una persona/cliente, expresa ante un programador.

Realizar los diagramas de clase es de suma importancia, esto permite llevar a cabo todas las funciones que se piensan para nuestra aplicación y posteriormente cosas que deberían implementarse o simplemente desecharse, en progreso con nuestra aplicación es de mucha ayuda a establecer los diferentes códigos implementados protegidos o públicos y como nuestras clases realizadas se va entre lanzando entre ella y hace más fácil poder entender el código realizado.

Establecer los requerimientos de confiabilidad y seguridad son importantes para proteger la relación con el cliente o usuario y darle la seguridad de que el programa funciona, no es de riesgo para nadie y las probabilidades de que falle son pocas.

Python de la mano con el módulo de pygame, permite una ilimitada creación de software debido a su amplia sintaxis y facilidad de comprensión gracias a la documentación.

Recomendaciones

Al utilizar Python, con la ayuda de Pycharm un editor de texto suele ser más sencillo entender la programación en el lenguaje que por medio de cualquier editor de texto.

Dividir bien el trabajo cuando se desarrolla un sistema es la clave para su realización a tiempo

Definir bien los requerimientos de un sistema maximiza las posibilidades de entregar un programa hecho correctamente y a la medida de un cliente.

Desarrollar los diferentes diagramas antes de realizar el código es de mucha ayuda antes de llegar al código y al igual que desarrollar los requerimientos de confiabilidad y seguridad son de mucha importancia para darle seguridad al usuario de que el programa funcionara correctamente.

Visualizar la documentación de pygame en su sitio web oficial ayuda mucho al trabajar con el módulo, ya que nos muestra la correcta utilización de los comandos disponibles.

Referencias Bibliográficas

https://es.wikipedia.org/wiki/Space_Invaders (Space Invaders) 03 de marzo de 2020

https://www.pygame.org/project/669 (Space Ship Game 1.4) 10 de marzo-Dylan J. Raub

https://www.pygame.org/docs/ (Pygame Front Page)

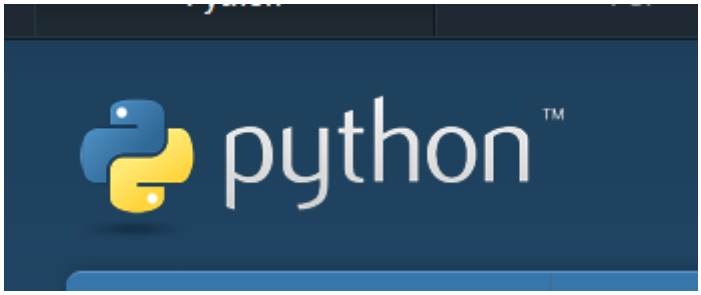
Manual de Usuarios

OBJETIVO

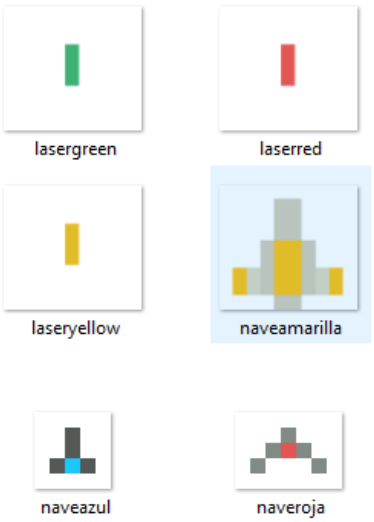
Dar a conocer la programación de un juego de 2D por medio de Python y un editor de texto

Requerimientos Necesarios

Para poder programar este juego se requiere de el lenguaje de programación Python y de un editor de texto esto es opcional , una serie de imágenes y el paquete de pygame , el juego es ejecutable en cualquier sistema operativo .

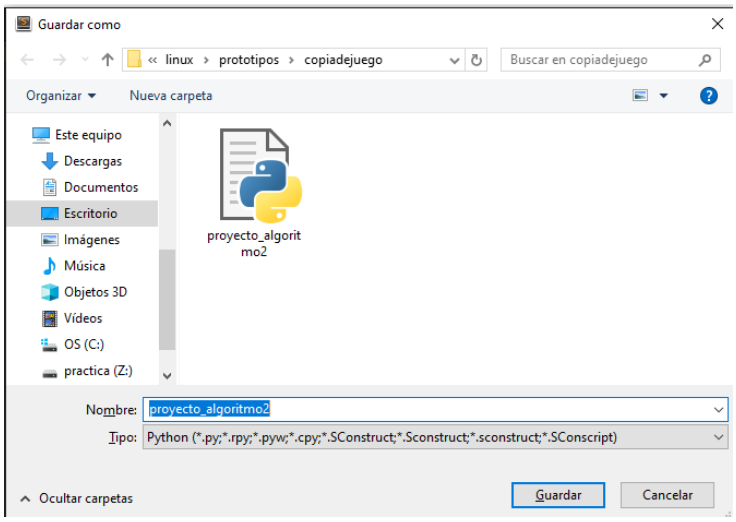


PYGAME



MANUAL DE PROGRAMACIÓN

Primero se le dará la extensión de .py para hacerle referencia que es un archivo con el lenguaje de python . El nombre del archivo tiene que ir junto no separado por espacios: ejemplo “proyecto.py “o “proyecto_fase1.py”.



Luego importamos lo módulos que ocuparemos con (**import**)

```
import pygame
import os
import time
import random
import sys
```

►**import pygame** =es un modulo donde podemos dibujar en la pantalla

►**import os** = nos permite manipular archivos de directorios .

► **import time** = este modulo nos permite trabajar con el tiempo e ejecución o movimiento que nosotros decidamos.

►**import random** = este modulo nos permite ver objetos de modo aleatorios en nuestra pantalla

Lo inicializamos con **pygame.font.init**

Luego escribimos dos variables para las medidas de la pantalla de nuestro programa **(ancho= x =300, alto= Y=300)**esto quiere decir que ancho me equivale a **Y** pero con una achura de 300 y alto es igual a **X** pero con una altura de 300. esto puede ser diferente a gusto de cada uno.

definimos una variable “ pantalla ” que sera igual **“ pygame.display.set_mode((ancho, alto)) “** quiere decir que pygame dibujanos una pantalla dentro del ordenador en estas mediadas.

Digitamos **“ pygame.display.set_caption** “**” Space Invader 2 ”** aquí estamos diciendo le a pygame que dibuja este texto como titulo dentro de la pantalla dibujada

```
pygame.font.init()

ancho,alto = 600,600
y =0
pantalla = pygame.display.set_mode((ancho,alto))
pygame.display.set_caption("Space invader 2")
```

ahora definimos variables para las imágenes que se utilizaran con **“ nombre de la variable = pygame.image.load (‘nombre del archivo con su extensión ‘)** aquí decimos a pygame que me dibuje la imagen de esta carpeta en pantalla.

```
#ENEMIGOS
NAVEROJA = pygame.image.load('naveroja.png')
NAVEAZUL = pygame.image.load('naveazul.png')
NAVEVERDE = pygame.image.load('naveverde.png')

#JUGADOR
NAVEAMARILLA = pygame.image.load('naveamarilla.png')

#DISPAROS
LASERED = pygame.image.load('laserred.png')
LASERBLUE = pygame.image.load('laserblue.png')
LASERGREEN = pygame.image.load('lasergreen.png')
LASERYELLOW = pygame.image.load('laseryellow.png')

FONDO =pygame.transform.scale(pygame.image.load('nuevo.png'), (ancho,alto))
MENU =pygame.transform.scale(pygame.image.load('menu.png'), (ancho,alto))
GOVER =pygame.transform.scale(pygame.image.load('gameover.png'), (ancho,alto))
TUTORIAL =pygame.transform.scale(pygame.image.load('tutorial.png'), (ancho,alto))
```

Pygame.transform.scale >>>(ancho,alto)

esto quiere decir que si el formato de la imagen es muy pequeña podemos ajustar a la pantalla con este comando .

definimos una variable donde se guarde otras variables que después ocuparemos

en este digitamos **“ def SpaceInvader2(): “** dentro de esta variable colocamos otra llamada **“ correr=True“** esta es para que el juego se ejecute y una variable llamada **“ FPS = 60 “** esta sera los fps a los que correrá el programa. A pygame

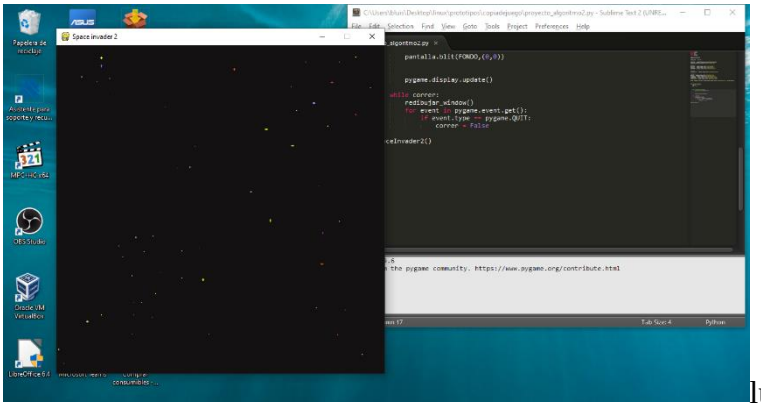
Y después digitamos **“ pygame.display.update() “** le decimos a python que nos actualice lo que tenemos .

```
29
30  def SpaceInvader2():
31      correr = True
32      FPS = 60
33
34
35      def redibujar_window():
36          pantalla.blit(FONDO,(0,0))
37
38
39          pygame.display.update()
40
41
```

Luego utilizando el ciclo **“ while “** iniciamos la variable correr junto con **“ redibujar_window “** para que me lo dibuje en pantalla para queme lo ejecute este sera el bucle del programa ,utilizando el comando **“ for “** le decimos que este atento a los eventos del programa que si no pasa nada lo detenga o permita finalizarlo .

```
39          pygame.display.update()
40
41      while correr:
42          redibujar_window()
43          for event in pygame.event.get():
44              if event.type == pygame.QUIT:
45                  correr = False
46
47  SpaceInvader2()
48
```

al final llamamos la variable **SpaceInvader2** para ejecutar . Nos enseñara una pantalla de este modo :



luego de eso digitaremos tres variables más dos para los niveles que serán **“ nivel = 0 y SpaceInvader2_font = pygame .font.SysFont (“comicsans”, 50) “** esta variable quiere decir que le estamos diciendo a pygame que escriba en el fondo del programa un sistema de configuración de fondo.

```
28
29
30  def SpaceInvader2():
31      correr = True
32      FPS = 60
33      nivel = 0
34      niveles = 5
35      SpaceInvader2_font = pygame.font.SysFont("comicsans",50)
36
```

Luego de esto digitamos en **redibujar_window()** la siguiente operación que seria **“ nivele_label = SpaceInvader2_font.render(f”nivele: {niveles}”, 1, (255,255,255) “**

Seguido de eso digitamos **“ pantalla.blit(nivele_label, (10,10))** para que nos dibuje el texto en la parte superior izquierda .

Así quedaría:

```
def redibujar_window():

    # pantalla.blit(FONDO,(0,0))
    #dibujar en pantalla (puntos , nivel , enemigos y perdio)
     nivel_label = SpaceInvader2_font.render(f"nivel : {nivel}",1 ,(255,255,255))
     puntos_label = SpaceInvader2_font.render(f"Puntos : {puntos}",1,(255,255,255))

     pantalla.blit(puntos_label, (10,10))
     pantalla.blit(nivel_label, (ancho - nivel_label.get_width() - 10, 10))
```

creamos dos variables llamadas **“ jugador = Jugador(400,400) “** el contenido de la variable debe ir con mayúscula la primera letra <<**Jugador**>> **Y la otra variable “ jugador_vel = 5 “** el primero es para poder crear y llamar la clase Jugador y la segunda sera la velocidad en la que se mueva la imagen en el programa y una llamada “ **clock = pygame.time.Clock() “** sera para representar el tiempo en el programa.

```
def SpaceInvader2():
    correr = True
    FPS = 60
    nivel = 0
    niveles = 5
    SpaceInvader2_font = pygame.font.SysFont("comicsans",50)

    jugador = Jugador(400,400)
    jugador_vel = 5
    clock = pygame.time.Clock()
```

después de eso crearemos la dos clases una llamada **Jugador** y otra llamada **Nave** porque Nave usaremos esta clase Nave para utilizarlas en la clase jugador y en la clase enemigos .

```
class Nave:
    def __init__(self, x, y, vida=100):
        self.x = x
        self.y = y
        self.vida = vida
        self.nave_img = None
        self.laser_img = None
        self.lasers = []
        self.cool_down_counter = 0

    def dibujar(self , pantalla):
        pantalla.blit(self.nave_img, (self.x, self.y))

    def get_ancho(self):
        return self.nave_img.get_width()

    def get_altura(self):
        return self.nave_img.get_height(
```

Clase Nave se utiliza **“ def __init__ (self, x ,y ,vida=100): “** definimos en método **init** para iniciar los atributos del objeto que creamos , self es para hacerle referencia ala clase , **el valor x es el ancho y el valor Y es la altura , vida=100** es que le estamos dando ala clase una vida de 100.

luego definimos **“ dibujar “**para que lo pueda dibujar en pantalla.

Definimos **“ get_ancho y get_alto “** para devolver el objeto .

Definimos la clase **Jugador** :

```
class Jugador(Nave):
    def __init__(self,x ,y, vida=100):
        super().__init__(x,y,vida)
        self.nave_img = NAVEAMARILLA
        self.laser_img = LASERYELLOW
        self.mask = pygame.mask.from_surface(self.nave_img)
        self.max_vida = vida


    def dibujar(self, pantalla):
        super().dibujar(pantalla)
```

la función **super()** nos permite invocar el atributo primario de la clase Nave

con **self. nave_img y laser_img** llamamos las imágenes que vamos a utilizar y con **self.max_vida** le estamos diciendo su máximo de vida y definimos otra variable de dibujar para ver lo en pantalla junto con la clase Nave .

Luego escribimos en la variable **redibujar_window()** donde le pediremos que nos muestre la clase en pantalla con **“ jugador.dibujar(pantalla).**

```
        pantalla.blit(niveles_label, (10,10))
        pantalla.blit(nivel_label, (ancho - nivel_label

        jugador.dibujar(pantalla)

        pygame.display.update()
```

Nos debe enseñar el programa de este modo .

Ahora le daremos movimiento a la nave con :

```
93                  correr = False
94
95          keys = pygame.key.get_pressed()
96          if keys[pygame.K_LEFT] and jugador.x + jugador_vel > 0: # izquierda
97              jugador.x -= jugador_vel
98          if keys[pygame.K_RIGHT] and jugador.x + jugador_vel + jugador.get_ancho() < ancho: #derecha
99              jugador.x += jugador_vel
100         if keys[pygame.K_UP] and jugador.y + jugador_vel > 0: # arriba
101             jugador.y -= jugador_vel
102         if keys[pygame.K_DOWN] and jugador.y + jugador_vel + jugador.get_altura() < alto: #abajo
103             jugador.y += jugador_vel
104
105
106
107 :Invader2()
108
```

aquí con **“ keys = pygame.key.get_pessed() “** estamos diciendo al software que este atento al presionar alguna tecla del keyboard (teclado)

con la función **if** estamos diciendo al pygame que cuando se presione la tecla de la flecha hacia a la izquierda que se mueva a la izquierda

con la función **if y el + jugador.get_ancho < ancho** le estamos diciendo que se mueva al lado contrario que seria la derecha.

Con esto ya tendríamos movimiento .

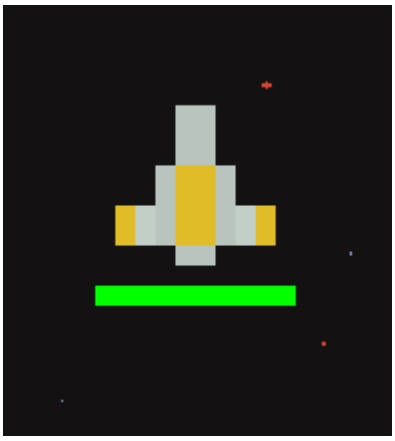
Luego agregaremos una barra de vida en la parte de abajo de la nave con el siguiente comando “

definimos una variable llamada barra en ella tendrá **“ pygame.draw.rect “** que significa pygame dibuja una linea recta en la pantalla de este color con una superficie y medias y la otra dice pygame dibujame otra linea sobre ella de este color cuando la vida vaya descendiendo.

Luego en dibujar digitamos **“ self.barra(pantalla) “** para que dibuje la variable .

```
        super().dibujar(pantalla)
        self.barra(pantalla)

    def barra(self, pantalla):
        pygame.draw.rect(pantalla, (255,0,0), (self.x, self.y + self.nave_img.get_height() + 10, self.nave_img.get_width(), 10))
        pygame.draw.rect(pantalla, (0,255,0), (self.x, self.y + self.nave_img.get_height() + 10, self.nave_img.get_width() * (self.vda/self.max_vida), 10))

class Enemigo(Nave):
```

será de este modo:



AHORA CLASE ENEMIGO

```
class Enemigo(Nave):
    COLOR_MAP = {
                    "red" : (NAVEROJA, LASERED),
                    "green": (NAVEVERDE, LASERGREEN),
                    "blue" : (NAVEAZUL, LASERBLUE)
                }
    def __init__(self, x, y, color, vida=100):
        super().__init__(x,y,vida)
        self.nave_img, self.laser_img = self.COLOR_MAP[color]
        self.mask = pygame.mask.from_surface(self.nave_img)

    def movimiento_enemigo(self, velocidad):
        self.y += velocidad
```

COLOR_MAP sera donde se guardara las imágenes de los enemigos que luego llamaremos , y **self.mask** esto quiere decir que dibujaremos la imagen una sobre otra pero en debido orden .

Definimos una variable de movimiento enemigo esto sera para cuando descienda los enemigos

definimos enemigos como una lista vaciá y la velocidad como **“ enemigo_vel”** junto con **“ rango = 5** “ y **“ laser_vel “**

```
def SpaceInvader2():
    correr = True
    FPS = 60
    nivel = 0
    niveles = 5
    SpaceInvader2_font = pygame.font.SysFont(
    rango = 5
    enemigos = []
    enemigo_vel = 1

    jugador = Jugador(400,400)
    jugador_vel = 5
    clock = pygame.time.Clock()
    laser_vel = 5
```

después de eso digitamos el comando **“ for “** en la variable **redibujar_window .**

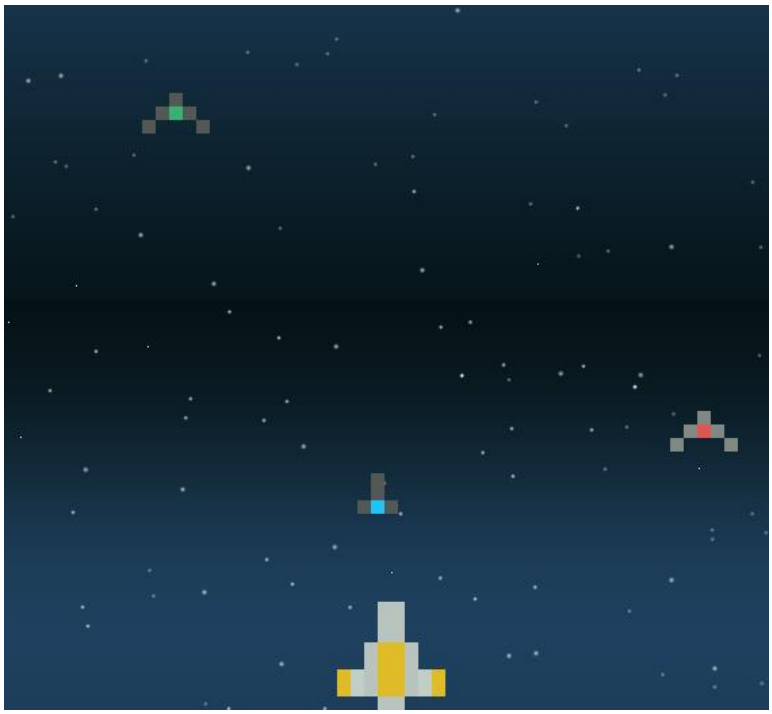
```
for Enemigo in enemigos:
    Enemigo.dibujar(pantalla)

jugador.dibujar(pantalla)
```

Luego con el comando **if** len queremos que los elementos de las lista los vaya aumentando y disminuyen en un rango determinado.

Con **range** le estoy diciendo que me enseñe los enemigos aleatorios dentro del programa.

```
if len(enemigos) == 0:
    nivel += 1
    rango += 5
    for i in range(rango):
        enemigo = Enemigo(random.randrange(50, ancho-100), random.randrange(-1500, -100), random.choice(["red","blue","green"]))
        enemigos.append(enemigo)
```

Este es el avance de programa :



Ahora la clase laser:

```
class laser:
    def __init__(self, x, y, img):
        self.x = x
        self.y = y
        self.img = img
        self.mask = pygame.mask.from_surface(self.img

    def dibujar(self, pantalla):
        pantalla.blit(self.img, (self.x,self.y))

    def movimiento(self, velocidad):
        self.y += velocidad

    def apagado(self, alto):
        return not (self.y <= alto and self.y >= 0)

class Nave:
```

ahora lo nuevo es que le agregamos una variable de apagado que quiere decir que cuando el objeto este a una determinada distancia se retorne con la función de **return** que regrese de nuevo y se cree.

Después en los comandos del **keys** al final le agregamos que este pendiente cuando se presione la tecla de **SPACE**

```
        jugador.y += jugador_vel
    if keys[pygame.K_SPACE]:
        jugador.disparo()
```

en la clase nave agregamos esta variable

```
class Nave:
    FRIO = 30
```

definimos el movimiento de laser hacia arriba de forma ascendente y una variable frio para que lleve un contador de disparo y la variable disparo para que proyecte el laser que es nuestra imagen.

```
    def mover_lasers(self,velocidad,objecto):
        self.frio()
        for laser in self.lasers:
            laser.movimiento(velocidad)
            if laser.apagado(alto):
                self.lasers.remove(laser)
    def frio(self):
        if self.cool_down_counter >= self.FRIO:
            self.cool_down_counter = 0
        elif self.cool_down_counter > 0:
            self.cool_down_counter += 1

    def disparo(self):
        if self.cool_down_counter == 0:
            laser = Laser(self.x,self.y,self.laser_img)
            self.lasers.append(laser)
            self.cool_down_counter = 1
```

En la clase Nave la variable dibujar digitaremos la condicional **For** laser esto sera para que nos dibuje el laser en la pantalla

```
def dibujar(self , pantalla):
    pantalla.blit(self.nave_img, (self.x, sel
    for laser in self.lasers:
        laser.dibujar(pantalla)

def mover_lasers(self,velocidad,objecto):
```

En la clase Jugador definiremos la variable mover_laser es la misma que esta en la clase Nave solo copiamos y pegamos. Esto es para que se puedan mover dentro de la clase Jugador con nuestro Jugador .

```
        self.max_vida = vida

    def mover_lasers(self,velocidad,objecto):
        self.frio()
        for laser in self.lasers:
            laser.movimiento(velocidad)
            if laser.apagado(alto):
                self.lasers.remove(laser)


    def dibujar(self, pantalla):
```

Después nos vamos hasta bajo antes de **SpaceInvader2()** y digitamos :

```
                enemigos.remove(enemigo)

        jugador.mover_lasers(-laser_vel, enemigos)



SpaceInvader2()
```

Regresamos a la clase Laser y al final escribimos una variable colisiones esto es para que laser nos detecte las colisiones ya sea con el jugador o con los enemigos .

```
        return not (self.y <= alto and self.y >= 0

    def colision(self, objecto):
        return chocar(self, objecto)
```

Y nos vamos a la variable **mover_lasers** en la clase Nave y digitamos el comando **elif** para si colisión choca con nave que le quite vida al objecto o los borre de la pantalla o lo remueva .

Ahora nos vamos a la clase Jugador y en la variable **mover_lasers** digitamos **else** con la condicional **for** si el

```
            laser.dibujar(pantalla)

    def mover_lasers(self,velocidad,objecto):
        self.frio()
        for laser in self.lasers:
            laser.movimiento(velocidad)
            if laser.apagado(alto):
                self.lasers.remove(laser)
            elif laser.colision(objecto):
                objecto.vida -= 10
                self.lasers.remove(laser)
```

objecto colisiona con objecto entonces quiero que remueva ese objecto en este caso el lasers cuando colisiones se remueve junto con la nave enemiga .

```
def mover_lasers(self,velocidad,objectos):
    self.frio()
    for laser in self.lasers:
        laser.movimiento(velocidad)
        if laser.apagado(alto):
            self.lasers.remove(laser)
        else:
            for objecto in objectos:
                if laser.colision(objecto):
                    objectos.remove(objecto)
                    self.lasers.remove(laser)
```

En la clase Enemigo definimos dos nuevas variables que son disparo y chocar
disparo que los lasers se agregan a la lista laser pero que se disparen
chocar que el objecto cuando choquen se enmascaren con otro objeto .

```
    def disparo(self):
        if self.cool_down_counter == 0:
            laser = Laser(self.x-20, self.y,self.laser_img)
            self.lasers.append(laser)
            self.cool_down_counter = 1

def chocar(objecto1 , objecto2):
    compensar_x = objecto2.x - objecto1.x
    compensar_y = objecto2.y - objecto1.y
    return objecto1.mask.overlap(objecto2.mask, (compensar_x, compensar_y)) != None
```

Luego digitamos los siguientes condicionales :

```
    #eventos de las colisiones
for enemigo in enemigos[:]:
    enemigo.movimiento_enemigo(enemigo_vel)
    enemigo.mover_lasers(laser_vel, jugador)

    if random.randrange(0, 2*60) == 1:
        enemigo.disparo()

    if chocar(enemigo, jugador):
        jugador.vida -= 10
        enemigos.remove(enemigo)
```

for enemigo = esto permite que las naves enemigas que vayan descendiendo puedan disparar aleatoriamente los lasers .

If random = permite que el enemigo dispare aleatorio en diferente tiempo .

If chocar = permite que la vida del jugador vaya descendiendo si las naves o los lasers enemigas toquen al jugador

definimos las ultimas variables **“ perdio_font = pygame.font.SysFont(comicsana , 60)**

perdio = False y perdio_contador = 0

```
def SpaceInvader2():
    correr = True
    FPS = 60
    nivel = 0
    niveles = 5
    SpaceInvader2_font = pygame.font.SysFont("comicsans",50)
    perdio_font = pygame.font.SysFont("comicsans", 60)
    rango = 5
    enemigos = []
    enemigo_vel = 1

    jugador = Jugador(400,400)
    jugador_vel = 5
    clock = pygame.time.Clock()
    laser_vel = 5
    perdio = False
    perdio_contador = 0
```

con la condicional **if perdió** le decimos que nos dibuje la imagen “perdió “ en la pantalla con los siguientes lo coloque en el centro de nuestra pantalla.

```
if perdio:
    gaover=pygame.mixer.Sound("gameover.ogg")
    gaover.play()
    gaover.set_volume(0.1)
    pantalla.blit(GOVER,(0,0))


pygame.display.update()
```

En el ciclo **While** con la condicional **if niveles** le decimos que si niveles es igual a 0 y la vida del jugador es igual o menor a 0 que me diga la variable **perdio = verdadera** o el contador de vidas vaya aumentando

y la condicional **if perdio** quiere decir que si mi jugador quedo sin barra de vida o se pasaron el limite de naves que automáticamente detenga mi programa pero si no que lo continué

```
while correr:
    clock.tick(FPS)
    redibujar_window()
    if niveles <= 0 or jugador.vida <= 0:
        perdio = True
        perdio_contador += 1

    if perdio:
        if perdio_contador > FPS * 3:
            correr = False
        else:
            continue
```

definimos una variable menú dentro de ella digitamos titulo y correr y otro ciclo while para que cargue el fondo nos muestre un titulo en el fondo que diga <<menu ,iniciar ,Tutorial…..>>quiere decir que con el mouse haga click para que corra el programa .

```
def menu():
    pygame.mixer.music.load("menu.ogg")
    pygame.mixer.music.play(-1)
    pygame.mixer.music.set_volume(0.03)
    titulo_font = pygame.font.SysFont("comicsans"
    correr = True
    while correr:
        pantalla.blit(MENU,(0,0))
        keys = pygame.key.get_pressed()
        if keys[pygame.K_r]:
            pantalla.blit(yano,(0,0))
        if keys[pygame.K_t]:
            pantalla.blit(TUTORIAL,(0,0))
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.mixer.stop()
                sys.exit()

       # if keys[pygame.K_SPACE]:
            #sw
        if keys[pygame.K_RETURN]:
            SpaceInvader2()
```

Anexos

Escala de evaluación

Evaluación	Muy bajo	Muy bien	Excelente
En cantidades	1-6	7-8	9-10

Autoevaluación.

Evaluación	Teoría	Programación	Puntualidad	Aporte a Grupo.
Carlos Mauricio	10	9	9	10
Bryan Vásquez	8	10	10	10

Coevaluación.

Coevaluación	Teoría	Código	Aporte
Carlos Mauricio	9	9	9
Bryan Vásquez	8	10	9