Revisión del Algoritmo MiniMax

Nombre: David Egas

Revisar la historia de los algoritmos (de acuerdo a lo planteado por Alan Turing para el caso del Minimax).

Historia

Alan Mathison Turing nació en Londres en 1912 y falleció en Wilmslow, Reino Unido en el año 1954 a la edad de 41 años, por envenenamiento de una manzana con cianuro. Turing fue quien desarrollo el primer programa de ordenador de la historia para jugar Ajedrez, esto se desarrolló a finales de los años 40, el cual desarrollo un artículo denominado Digital Computers Applied to Games.

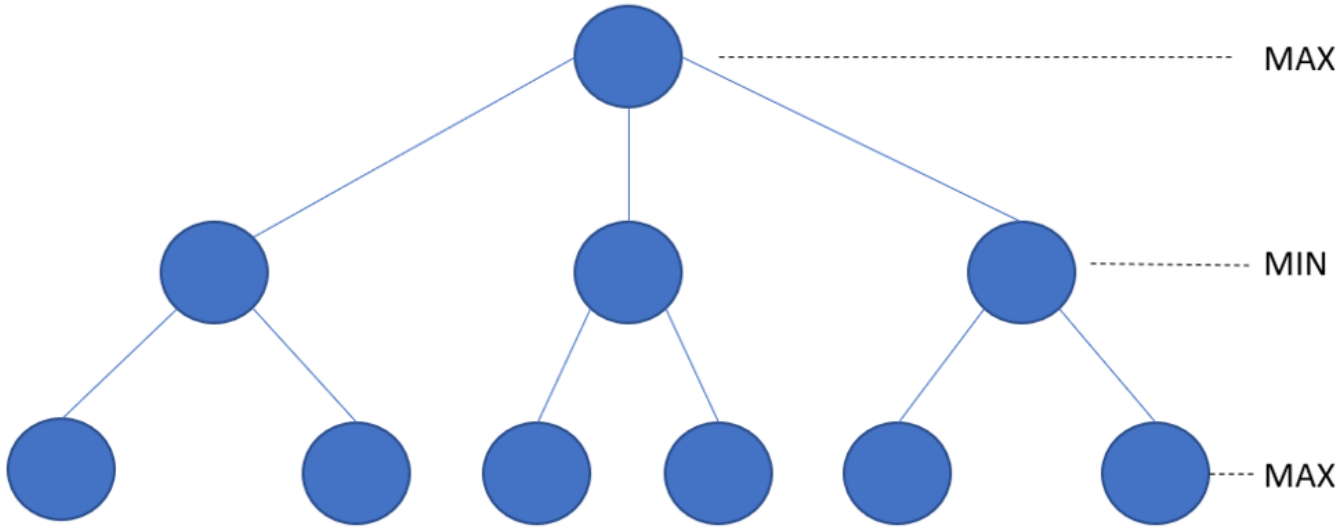
Turing demostró que había problemas irresolubles, es decir, sin solución algorítmica. Para dar forma al concepto ideó la famosa máquina que lleva su nombre, un dispositivo imaginario que, una vez construido, podría ejecutar cualquier operación matemática resoluble por medio de un algoritmo, y que, en el caso de programarse, se transformaría en un ordenador. Pero Turing jamás llegó a materializar su proyecto, al no contar con los medios técnicos necesarios.

Algoritmo Minimax

El algoritmo de búsqueda Minimax, es una búsqueda de profundidad limitada. El nombre del algoritmo se da al considerar que dada una función estática que devuelve valores en relación al jugador maximizante, este trata de maximizar su valor mientras que el oponente trata de minimizarlo. En un árbol de juego donde los valores de la función estática, están en relación con el jugador maximizante, se maximiza y minimiza de forma alterna de un nivel a otro.

En teoría de juegos, Minimax es un algoritmo de decisión para tratar de minimizar la perdida máxima, que se espera en los juegos que contienen adversarios. El funcionamiento de Minimax se define a como se puede elegir el mejor movimiento, dado el caso que el contrincante escoja el peor movimiento.

Estructura del algoritmo minimax:

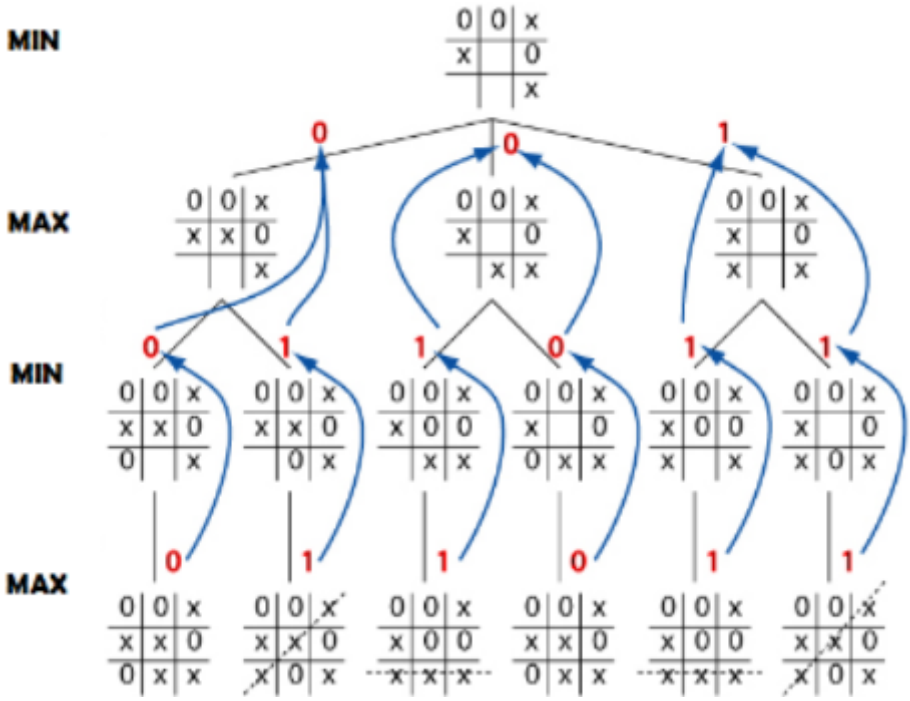


Explicar mediante un ejemplo práctico el funcionamiento del algoritmo.

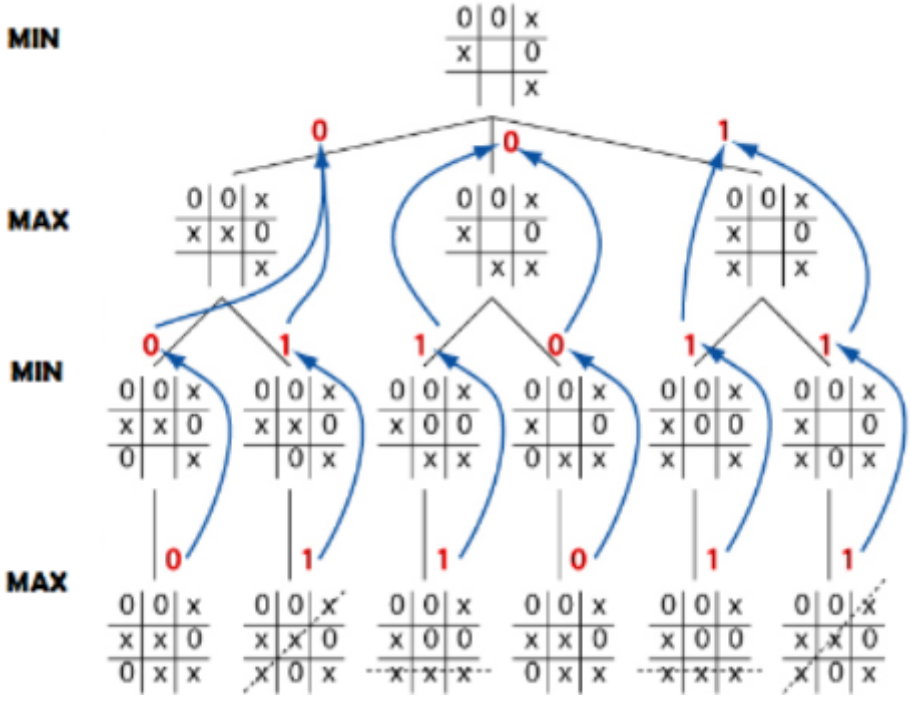
El algoritmo minimax busca que la IA decida el mejor movimiento o el que le perjudique menos, teniendo en cuenta que el contrincante buscara su mejor movimiento.

El ejemplo a continuación es un tres en raya:

1. Es una configuración inicial del juego, es decir, un estado en el que se encuentre el juego. Para nuestro ejemplo sería:

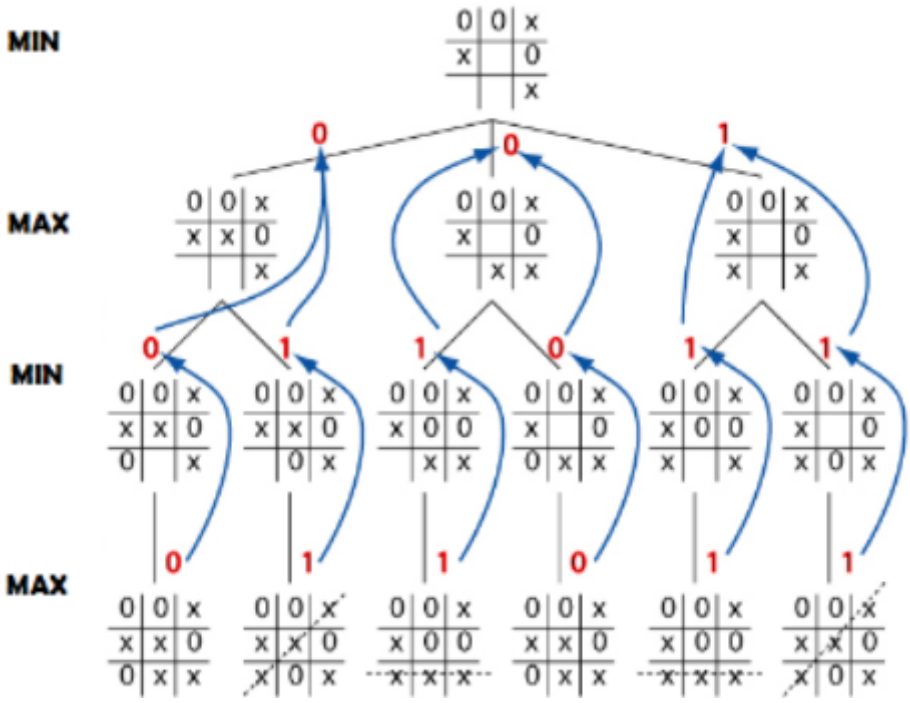


1. Operadores: Corresponden a las jugadas legales que se pueden hacer en el juego, en el caso del tres en raya no puedes marcar una casilla ya antes marcada.

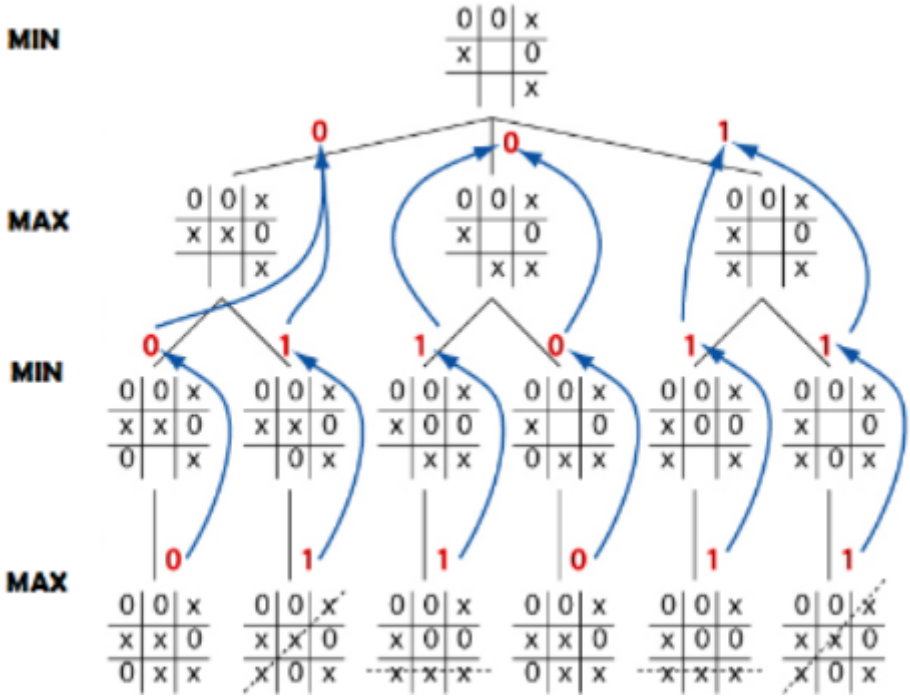


1. Condición Terminal: Determina cuando el juego se acabó, en nuestro ejemplo el juego termina cuando un jugador marca tres casillas seguidas iguales, ya se horizontalmente, verticalmente o en diagonal, o

se marcan todas las casillas (empate) .

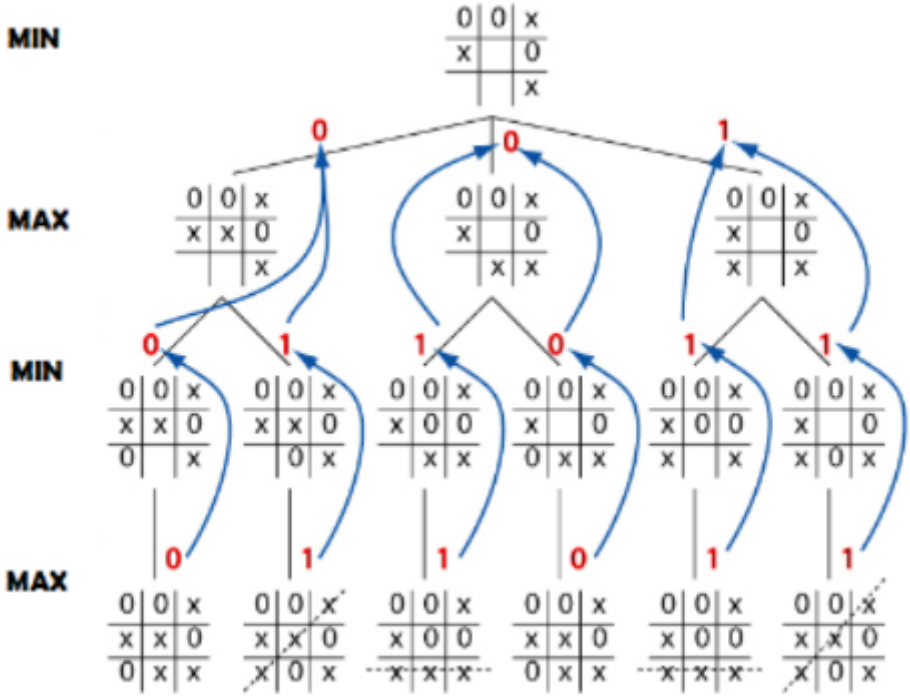


1. Función de Utilidad: Da un valor numérico a una configuración final de un juego. En un juego en donde se puede ganar, perder o empatar, los valores pueden ser 1, 0, o -1.

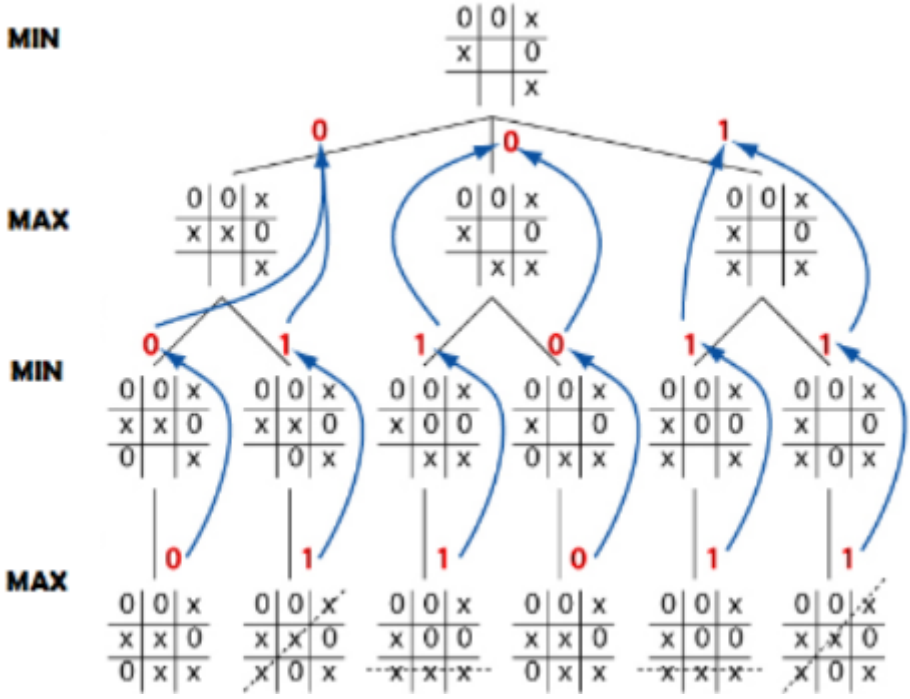


Implementación Minimax: Los pasos que sigue minimax pueden variar, pero lo importante es tener una idea clara de cómo es su funcionamiento, los pasos a seguir son:

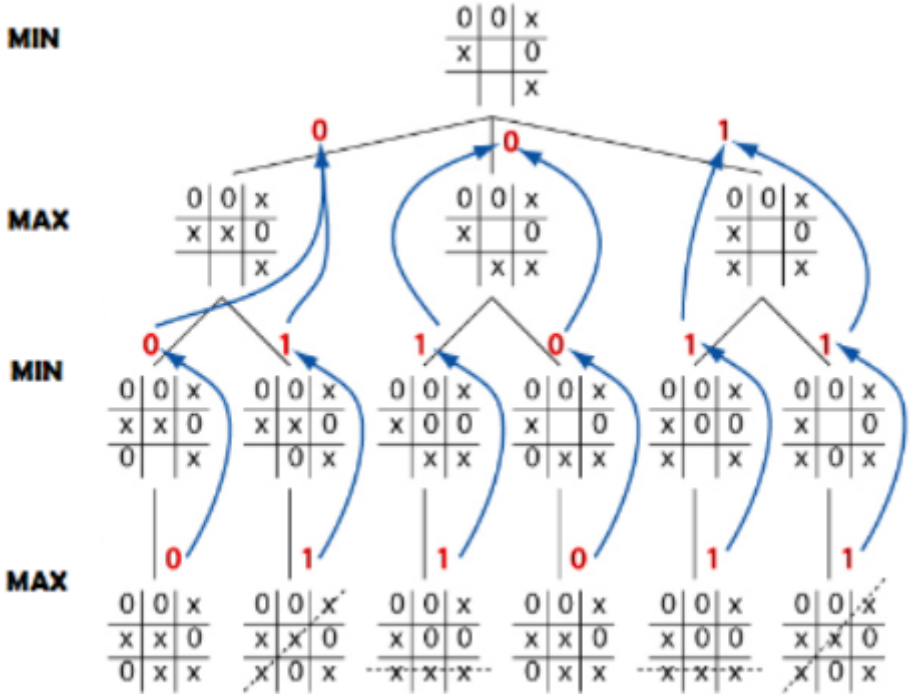
1. El algoritmo primero generar un árbol de soluciones completo a partir de un nodo dado. veamos el siguiente ejemplo:



1. Para cada nodo final, buscamos la función de utilidad de estos. En nuestro ejemplo usaremos un 0 para las partidas que terminen en empate, un 1 para las que gane la IA y un -1 para las que gane el jugador humano.



1. Y lo que hará el algoritmo Minimax cuando vaya regresando hacia atrás, será comunicarle a la llamada recursiva superior cuál es el mejor nodo hoja alcanzado hasta el momento. Cada llamada recursiva tiene que saber a quién le toca jugar, para analizar si el movimiento realizado pertenece a la IA o al otro jugador, ya que cuando sea el turno de la IA nos interesa MAXIMIZAR el resultado, y cuando sea el turno del rival MINIMIZAR su resultado.



Espero que les haya gustado este tutorial y les pueda servir para que desarrollen sus propios juegos, no olviden que si tienen dudas y consultas, pueden dejaras en los comentarios. Hasta la próxima.

Realizar una propuesta de cómo emplear el algoritmo en su problema del laberinto.

Este algoritmo se puede aplicar al problema del laberinto para encontrar los movimientos optimos que nos permitan encontrar la salida del mismo, se genera un arbol con todos los posibles movimientos y se da un valor de 0 al movimiento que nos lleve a una pared y un 1 para el camino sin pared, este metodo se ejecutará de manera recursiva para poder llegar a la salida.

Revisar al menos 4 papers científicos de aplicaciones prácticas de este algoritmo y su variante.

1. Stockman, G. C. (1979). A minimax algorithm better than alpha-beta?. Artificial Intelligence, 12(2), 179-196.
2. Vardi, A. (1992). New minimax algorithm. Journal of Optimization Theory and Applications, 75(3), 613-634.
3. Takimoto, E., & Warmuth, M. K. (2000, December). The last-step minimax algorithm. In International Conference on Algorithmic Learning Theory (pp. 279-290). Springer, Berlin, Heidelberg.
4. Borovska, P., & Lazarova, M. (2007, June). Efficiency of parallel minimax algorithm for game tree search. In Proceedings of the 2007 international conference on Computer systems and technologies (pp. 1-6).

Juego de naves

Juego donde una nave dispara a unos asteroides que se acumulan según el tiempo avanza, el juego aplica fisicas de movimiento para la nave y los asteroides, muestra los puntajes y la vida del jugador.

Librerías usadas:

- Librería simplegui
- Librería math
- Librería random

In [2]:

```
#Este try/catch sirve para corregir un error al momento de implementar la librería s
implegui
#Es parte de la investigación el corregir este error para la tarea de Inteligencia A
rtificial
try:
    import simplegui
except ImportError:
    import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
import math
import random

class ImageInfo:
    def __init__(self, center, size, radius = 0, lifespan = None, animated = False):
        self.center = center
        self.size = size
        self.radius = radius
        if lifespan:
            self.lifespan = lifespan
        else:
            self.lifespan = float('inf')
        self.animated = animated

    def get_center(self):
        return self.center

    def get_size(self):
        return self.size

    def get_radius(self):
        return self.radius

    def get_lifespan(self):
        return self.lifespan

    def get_animated(self):
        return self.animated


debris_info = ImageInfo([320, 240], [640, 480])
debris_image = simplegui.load_image("http://commondatastorage.googleapis.com/codesku
lptor-assets/lathrop/debris2_blue.png")
nebula_info = ImageInfo([400, 300], [800, 600])
nebula_image = simplegui.load_image("")
splash_info = ImageInfo([200, 150], [400, 300])
splash_image = simplegui.load_image("http://commondatastorage.googleapis.com/codesku
lptor-assets/lathrop/splash.png")
ship_info = ImageInfo([45, 45], [90, 90], 35)
ship_image = simplegui.load_image("http://commondatastorage.googleapis.com/codeskulp
tor-assets/lathrop/double_ship.png")
missile_info = ImageInfo([5,5], [10, 10], 3, 50)
missile_image = simplegui.load_image("http://commondatastorage.googleapis.com/codesk
ulptor-assets/lathrop/shot2.png")
asteroid_info = ImageInfo([45, 45], [90, 90], 40)
asteroid_image = simplegui.load_image("http://commondatastorage.googleapis.com/codes
kulptor-assets/lathrop/asteroid_blue.png")
explosion_info = ImageInfo([64, 64], [128, 128], 17, 24, True)
explosion_image = simplegui.load_image("http://commondatastorage.googleapis.com/code
skulptor-assets/lathrop/explosion_alpha.png")
soundtrack = simplegui.load_sound("http://commondatastorage.googleapis.com/codeskulp
```

```
tor-assets/sounddogs/soundtrack.mp3")
missile_sound = simplegui.load_sound("http://commondatastorage.googleapis.com/codesk
ulptor-assets/sounddogs/missile.mp3")
ship_thrust_sound = simplegui.load_sound("http://commondatastorage.googleapis.com/co
deskulptor-assets/sounddogs/thrust.mp3")
explosion_sound = simplegui.load_sound("http://commondatastorage.googleapis.com/code
skulptor-assets/sounddogs/explosion.mp3")



# globals for user interface
WIDTH = 800
HEIGHT = 600
score = 0
lives = 3
time = 0
started = False

def angle_to_vector(ang):
    return [math.cos(ang), math.sin(ang)]

def dist(p, q):
    return math.sqrt((p[0] - q[0]) ** 2 + (p[1] - q[1]) ** 2)

def process_sprite_group(s_group, canvas):
    for x in set(s_group):
        x.draw(canvas)
        if x.update():
            s_group.remove(x)


def group_collide(s_group, other_object):
    has_collide = False

    for x in set(s_group):
        if (x.collide(other_object)):
            a_explosion = Sprite(x.get_position(), [0, 0], 0, 0, explosion_image, ex
plosion_info, explosion_sound)
            explosion_group.add(a_explosion)
            s_group.remove(x)
            has_collide = True

    return has_collide


def group_group_collide(group1, group2):
    total_collisions = 0

    for x in set(group1):
        if group_collide(group2, x):
            group1.discard(x)
            total_collisions += 1

    return total_collisions

class Ship:

    def __init__(self, pos, vel, angle, image, info):
        self.pos = [pos[0], pos[1]]
        self.vel = [vel[0], vel[1]]
        self.thrust = False
```

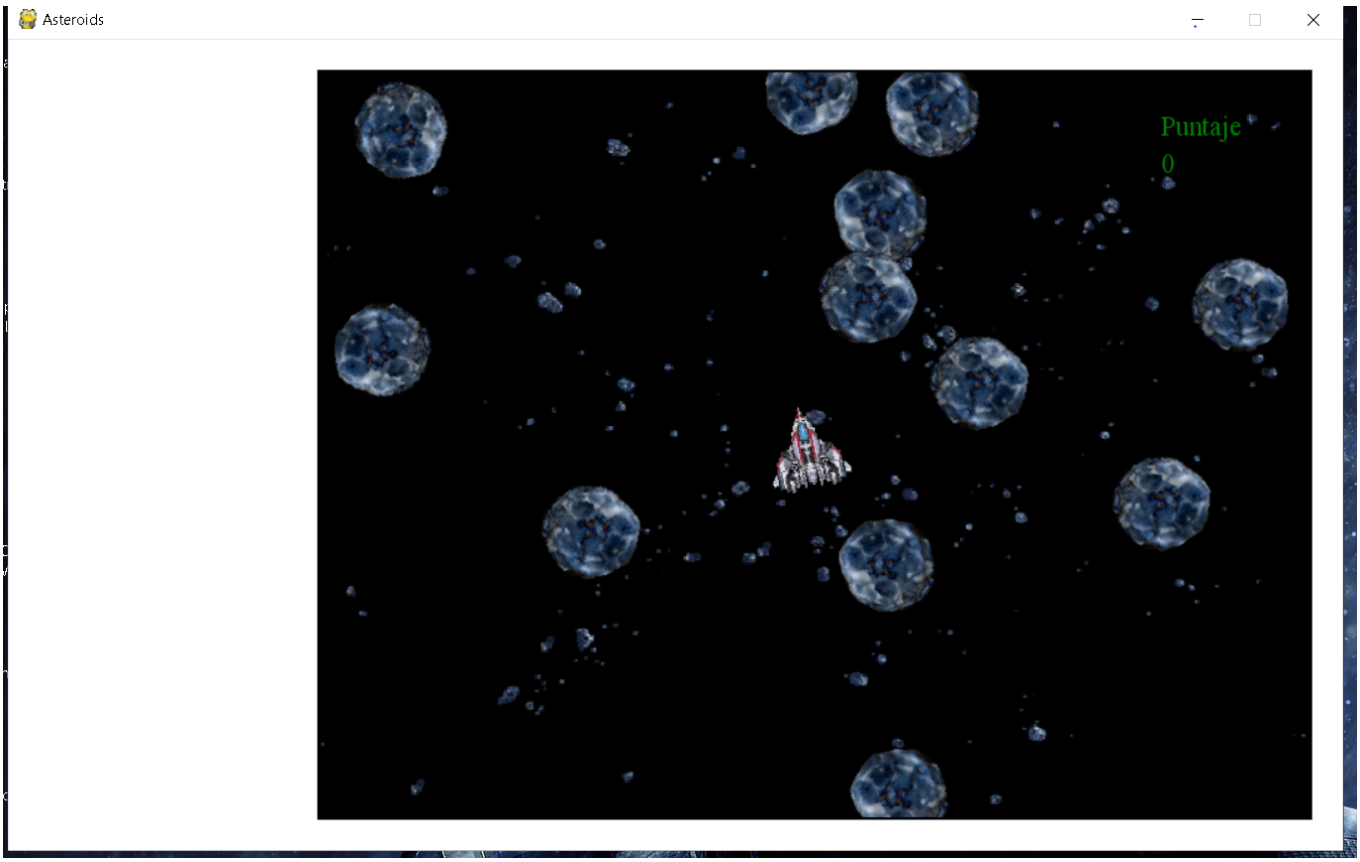
```
        self.angle = angle
        self.angle_vel = 0
        self.image = image
        self.image_center = info.get_center()
        self.image_size = info.get_size()
        self.radius = info.get_radius()

    def get_position(self):
        return self.pos

    def get_radius(self):
        return self.radius

    def draw(self,canvas):
        if self.thrust:
            canvas.draw_image(self.image, [self.image_center[0] + self.image_size[0
], self.image_center[1]] , self.image_size,
                              self.pos, self.image_size, self.angle)
        else:
            canvas.draw_image(self.image, self.image_center, self.image_size,
                              self.pos, self.image_size, self.angle)

    def update(self):

        self.angle += self.angle_vel

        self.pos[0] = (self.pos[0] + self.vel[0]) % WIDTH
        self.pos[1] = (self.pos[1] + self.vel[1]) % HEIGHT

        if self.thrust:
            acc = angle_to_vector(self.angle)
            self.vel[0] += acc[0] * .1
            self.vel[1] += acc[1] * .1

        self.vel[0] *= .99
        self.vel[1] *= .99

    def set_thrust(self, on):
        self.thrust = on
        if on:
            ship_thrust_sound.rewind()
            ship_thrust_sound.play()
        else:
            ship_thrust_sound.pause()

    def increment_angle_vel(self):
        self.angle_vel += .05

    def decrement_angle_vel(self):
        self.angle_vel -= .05

    def shoot(self):
        forward = angle_to_vector(self.angle)
        missile_pos = [self.pos[0] + self.radius * forward[0], self.pos[1] + self.ra
dius * forward[1]]
        missile_vel = [self.vel[0] + 6 * forward[0], self.vel[1] + 6 * forward[1]]
        a_missile = Sprite(missile_pos, missile_vel, self.angle, 0, missile_image, m
issile_info, missile_sound)

        missile_group.add(a_missile)
```

```
class Sprite:
    def __init__(self, pos, vel, ang, ang_vel, image, info, sound = None):
        self.pos = [pos[0],pos[1]]
        self.vel = [vel[0],vel[1]]
        self.angle = ang
        self.angle_vel = ang_vel
        self.image = image
        self.image_center = info.get_center()
        self.image_size = info.get_size()
        self.radius = info.get_radius()
        self.lifespan = info.get_lifespan()
        self.animated = info.get_animated()
        self.age = 0
        if sound:
            sound.rewind()
            sound.play()

    def get_position(self):
        return self.pos

    def get_radius(self):
        return self.radius

    def draw(self, canvas):
        if self.animated:
            canvas.draw_image(self.image, [self.image_center[0] + self.age*self.imag
e_size[0] ,self.image_center[1]] , self.image_size,
                              self.pos, self.image_size, self.angle)
        else:
            canvas.draw_image(self.image, self.image_center, self.image_size,
                              self.pos, self.image_size, self.angle)

    def update(self):
        # update angle
        self.angle += self.angle_vel

        # update position
        self.pos[0] = (self.pos[0] + self.vel[0]) % WIDTH
        self.pos[1] = (self.pos[1] + self.vel[1]) % HEIGHT

        self.age += 1
        return (self.age >= self.lifespan)


    def collide(self,other_object):
        return ( dist(self.get_position(), other_object.get_position()) <= (self.rad
ius + other_object.radius) )

def keydown(key):
    if key == simplegui.KEY_MAP['left']:
        my_ship.decrement_angle_vel()
    elif key == simplegui.KEY_MAP['right']:
        my_ship.increment_angle_vel()
    elif key == simplegui.KEY_MAP['up']:
        my_ship.set_thrust(True)
    elif key == simplegui.KEY_MAP['space']:
        my_ship.shoot()

def keyup(key):
    if key == simplegui.KEY_MAP['left']:
        my_ship.increment_angle_vel()
```

```
    elif key == simplegui.KEY_MAP['right']:
        my_ship.decrement_angle_vel()
    elif key == simplegui.KEY_MAP['up']:
        my_ship.set_thrust(False)

def click(pos):
    my_ship.shoot()
    global started, lives, score
    center = [WIDTH / 2, HEIGHT / 2]
    size = splash_info.get_size()
    inwidth = (center[0] - size[0] / 2) < pos[0] < (center[0] + size[0] / 2)
    inheight = (center[1] - size[1] / 2) < pos[1] < (center[1] + size[1] / 2)
    if (not started) and inwidth and inheight:
        started = True
        lives = 3
        score = 0
        soundtrack.rewind()
        soundtrack.play()

def draw(canvas):
    global time, started, lives, score, rock_group


    time += 1
    wtime = (time / 4) % WIDTH
    center = debris_info.get_center()
    size = debris_info.get_size()
    canvas.draw_image(nebula_image, nebula_info.get_center(), nebula_info.get_size
(), [WIDTH / 2, HEIGHT / 2], [WIDTH, HEIGHT])
    canvas.draw_image(debris_image, center, size, (wtime - WIDTH / 2, HEIGHT / 2), (
WIDTH, HEIGHT))
    canvas.draw_image(debris_image, center, size, (wtime + WIDTH / 2, HEIGHT / 2), (
WIDTH, HEIGHT))

    canvas.draw_text("Vidas", [50, 50], 22, "Green")
    canvas.draw_text("Puntaje", [680, 50], 22, "Green")
    canvas.draw_text(str(lives), [50, 80], 22, "Green")
    canvas.draw_text(str(score), [680, 80], 22, "Green")

    my_ship.draw(canvas)
    my_ship.update()

    process_sprite_group(rock_group,canvas)
    process_sprite_group(missile_group,canvas)
    process_sprite_group(explosion_group,canvas)

    if group_collide(rock_group, my_ship):
        lives -= 1

    score += group_group_collide(rock_group, missile_group)

    if lives == 0:
        started = False
        rock_group = set()

    if not started:
        canvas.draw_image(splash_image, splash_info.get_center(),
                          splash_info.get_size(), [WIDTH / 2, HEIGHT / 2],
                          splash_info.get_size())

def rock_spawner():
```

```
    if started:
        if len(rock_group) != 12:
            rock_pos = [random.randrange(0, WIDTH), random.randrange(0, HEIGHT)]

            if not (dist(my_ship.get_position(), rock_pos) <= 10):
                rock_vel = [(random.random() * .6 - .3) + score/5.0 , (random.random
() * .6 - .3) + score/5.0]
                rock_avel = random.random() * .2 - .1
                a_rock = Sprite(rock_pos, rock_vel, 0, rock_avel, asteroid_image, as
teroid_info)

                rock_group.add(a_rock)

frame = simplegui.create_frame("Asteroids", WIDTH, HEIGHT)

my_ship = Ship([WIDTH / 2, HEIGHT / 2], [0, 0], 0, ship_image, ship_info)
rock_group = set([])
missile_group = set([])
explosion_group = set([])

frame.set_keyup_handler(keyup)
frame.set_keydown_handler(keydown)
frame.set_mouseclick_handler(click)
frame.set_draw_handler(draw)

timer = simplegui.create_timer(1000.0, rock_spawner)

timer.start()
frame.start()
```

Resultados



In [ ]:

```

```