ESTÁNDAR DE CODIFICACIÓN

MIGUEL ÁNGEL CASTILLO SÁNCHEZ

14/09/2016

Table of Contents

Convenciones Generales

Nombres

Todos lo nombres deberán ser escritos en inglés, los nombres deberán ser lo más corto posibles para evitar nombres de variables demasiados largos, los nombres de las variables deben de ser descriptivos.

El nombre de las funciones deberá de representar una acción, así que deben de comenzar con un verbo.

El nombre de las variables deberá de ser un sustantivo ya que representa un atributo de la clase o de la función.

Línea

El número de caracteres máximos por línea es de 80.

Estructura de Clases

Cada archivo que se cree será una clase, el nombre del archivo deberá relacionarse con el objeto que representa. Los nombres de las clases deberán de seguir el siguiente formato:

Ejemplo:

```
player.gd

cacao_item.gd
```

En la parte superior de los archivo de clase deberá de tener la siguiente leyenda:

```
## Copyright 2016 Yakanda Studios | Mantra
## Naktan - Mantra por especificar
```

Deberán de aparecer los componentes de la clase en el siguiente orden, y colocar las variables y funciones que se relacionan juntas:

```
## exports public variables
export(int) var life = 2 setget set_life, get_life

## private variables
var direction
var origin setget ,get_origin

## constantes
const IDLE = 1

## Flags
var FLAG_ACTIVE setget set_flag_active, is_active
## signals
signal loosing_life

##virtual functions override
func _init():
	pass
func _fixed_process(delta):
	pass

## behaviour functions
func speed_up(delta):
	pass

## setters y getters
func set_life(init_life):
	pass
```

Exports

Las variables exports son variables públicas que se pueden acceder vía setter o getter si está habilitado. Estas variables definen un tipo, además deberán de ir inicializadas según la documentación.

Ejemplo:

```
export(int) var life = 3 setget set_life, get_life
```

Variables

Los nombres de las variables deberán ser descriptivos en cuanto al valor que resguardan, de tal forma que al leerla se entenderá su propósito en el código. Si la variable necesita de dos palabras estas deberán ser separadas por un guion bajo, todas las variables deberán ser escritas en minúsculas.

Ejemplo:

```
var message

var secrete_message
```

Constantes

Las constantes son valores que no cambian a través del tiempo. Deberán de ser escritas usando mayúsculas. Las constantes deberán tener nombres descriptivo del valor que resguardan, si el nombre esta descrito por más de dos palabras deberán de ser separadas por un guion bajo.

Ejemplo:

```
const MAX_LIVES = 5
```

Banderas

Las banderas representan algún comportamiento especial que se puede activar o desactivar. El nombre de las banderas deberá de comenzar con la palabra FLAG_ seguida del nombre de la bandera.

En el caso del getter deberá de comenzar el nombre con is_ seguido del nombre de la bandera sin la palabra flag.

Ejemplo:

```
var FLAG_ACTIVE setget set_flag_active, is_active
```

Señales

Las señales deberán tener nombres que inicien con un verbo y con el nombre de la variable que vigilan o el estado que notifican.

Ejemplo:

```
signal loosing_life
```

Métodos

El nombre de los métodos debe de ser descriptivo para saber lo que hace el método en un solo vistazo, por los cual los métodos deben de ir acompañados de un verbo conjugado, haciendo referencia a la acción que va a realizar el método, si el nombre del método requiere de más de dos palabras deberán ser separados por un guion bajo. No se deberá de dejar un espacio antes del paréntesis que abre.

Ejemplo:

```
func draw_level():
```

El nombre de las funciones para recibir las señales deberá de empezar con la palabra on_ seguida del nombre de la señal a cachar.

Ejemplo:

```
func on_loosing_life():
      pass
func on_loosing_mana(new_mana, time_elapsed, args):
```

Setters - Getters

Las variables que necesiten ser modificadas o vistas desde fuera de la clase deberán de tener configurado un setter o getter, en caso de sólo necesitar el valor de la variable sólo se deberá de asignar un getter, si sólo se necesita modificar el valor entonces sólo se deberá de colocar un setter.

El nombre de los métodos siempre deberán de empezar con get o set según sea el caso seguido de un guion bajo y nombre de la variable a la que están configurando.

En cada uno de los métodos se puede colocar el procedimiento que mejor convenga para ver la variable o para asignar un valor.

La variable que recibe el setter deberá de tener un nombre que haga referencia al cambio que va a realizar en la variable a modificar, generalmente puede ser como *new_value*, o *init_value*, dependiendo el caso.

Ejemplo:

```
#configuración de settter y getter
var score setget set_score,get_score

#configuración de setter
```

```
var score setget set_score

#configuración de getter
var score setget ,get_score

#métodos set / get
func set_score(new_score):
score = new_score

func get_score():
      return score
```

Identación

Se ocupará identación de 4 espacios, al ser gdscript un lenguaje que no ocupa llaves para separar los bloques de código, se deberá de tener cuidado al usar la ideación ya que todo elemento que este dentado estará dentro del elemento que no este identado.

Ejemplo:

```
#erroneo
func draw_something():
      var axis_x
            var axis_y
#correcto
func draw_something():
      var axis_x
      var axis_y
```

Operaciones

Las operaciones entre variables se deberán de colocar dejando un espacio entre los operadores, no así para los paréntesis, se usarán paréntesis en caso de tener más de tres elementos en la operación. También se deberá de dejar un espacio entre el igual y el nombre de la variable.

Ejemplo:

```
var res_x = axis_x - axis_y
```

IF

Las expresiones contenidas en la comparación NO deberán de llevar paréntesis cuando se trate de 2 variables, se deberá usar la menor

cantidad de IF posible. Se deberá de dejar un espacio entre el if y paréntesis o de la condición que se está construyendo.

Ejemplo:

```
#correcto
if axis_x > axis_y:


#incorrecto
if (axis_x > axis_y):


#en caso de tener más condiciones
if ((axis_x > axis_y) and (axis_z < axis_x)):
```

Ciclos

En el caso del **while** se usará lo mismo descrito en los if, en el caso de for se usarán nombres descriptivos en los índices que se utilizan para saber que está recorriendo el índice.

Ejemplo:

```
for item in coleccion:
      do something …
```

Documentación

La documentación dentro del código(dentro del cuerpo de las funciones), se deberá de hacer sólo cuando sea necesario describir alguna funcionalidad. En el caso de las funciones de comportamiento deberán de documentarse en todo momento, siendo directos en lo que realiza la función. También deberán de tener #TODO para las funcionalidad faltantes o sugerencias a futuro.

Comentarios de una línea

Estos comentarios se usarán para describir alguna acción dentro del cuerpo de la función en caso de ser necesario para aclarar ciertos pasos, estos comentarios deberán de usar ## seguido de un espacio.

Ejemplo:

```
func draw_level():
      ## variable de eje x
      var axis_x
```

Comentarios de Funciones

Los comentarios deberán ser lo más breves posibles, describiendo lo que hace el método, si el método devuelve una valor deberá de contener un @return seguido del tipo de la variable de retorno, en el

caso de los argumentos se deberá de usar @param seguida con el tipo de la variable su nombre y una descripción si es necesaria.

Ejemplo:

```
# Dibuja un nivel
func draw_level():
	pass


# Calcula la habilidad del jugador con base a sus
estadísticas
# @return int habilidad del jugador
func calculate_player_hability():
	pass


# Calcula la habilidad del jugador con base a sus
estadísticas
#
# @param int score | score del jugador
# @param int lives | vidas del jugador
# @return int habilidad | del jugador
func calculate_player_hability(score, lives):


Se pueden hacer divisiones de operaciones de código


#### Segmento de código ###
```