Memoria de Título: Prácticas de Construcción y Optimización de Código en la Plataforma PICO-8

Milenko Bonacich Saldias

Carrera: Ingeniería civil informática.

Profesor Patrocinante: Geoffrey Hecht

2024.

Resumen

PICO-8 es una plataforma de desarrollo primordialmente de videojuegos que ha atraído una considerable cantidad de personas con su filosofía de proyectos pequeños, simulando limitaciones de décadas pasadas en las cuales el tamaño de un programa era muy limitado debido a la poca cantidad de memoria tanto en computadores como discos. La comunidad es vibrante, desarrollando proyectos de toda índole además de videojuegos. En sí, la plataforma es un ambiente completo y minimalista de desarrollo que puede ser útil para la creación de prototipos, enseñanza de programación o simplemente para aficionados. Las limitaciones impuestas, aunque sean parte de la atracción, son murallas que pueden ser escaladas con apropiadas prácticas de programación. Utilizando estas prácticas podemos mejorar la utilización de recursos del código, pero usualmente esto tiene como costo la legibilidad de este; es importante balancear recursos y legibilidad para obtener lo mejor de ambos mundos y así desarrollar un proyecto capaz de realizar todo lo planeado por el programador.

Tomando un set de 2966 proyectos curados por la comunidad, de múltiples categorías, tanto juegos y herramientas creadas en la plataforma, se analizaron ciertos rasgos de la utilización de recursos: La compresión y caracteres están conectados intrínsicamente, y raramente alcanzan el límite, mientras que los tokens son independientes y es más ocurrente que lleguen a los altos percentiles del límite. Utilizando lo estudiado a lo largo del informe, se realizó el estudio de un proyecto por el cual se tomaron varias prácticas, discutiéndolas en contraste con lo hallado en el código y concluyendo en como el proyecto puede ser mejorado, lo cual puede ser utilizado como base para la evaluación de otros proyectos en la plataforma PICO-8.

Summary

PICO-8 is a primarily game development platform that has attracted a considerable number of people with its philosophy of small projects, simulating limitations from past decades when program size was very restricted due to limited memory on computers and disks. The community is vibrant, developing projects of all kinds in addition to video games. Overall, the platform is a complete and minimalist environment that may be useful for prototyping, teaching programming, or simply for enthusiasts. The imposed limitations, although part of the attraction, are walls that can be scaled with appropriate programming practices. By using these practices, we can improve resource utilization in the code, but this usually comes at the cost of readability; it is important to balance resources and readability to get the best of both worlds and thus develop a project capable of achieving everything the programmer planned.

Taking a set of 2,966 projects curated by the community, from multiple categories, including games and tools created on the platform, certain traits of resource utilization were analyzed: compression and characters are intrinsically connected and rarely reach the limit, while tokens are independent, and it is more common for them to reach the higher percentiles of the limit. Using what was studied throughout the report, a case study was conducted on a project, from which various practices were taken, discussing them in contrast to what was found in the code and concluding how the project can be improved, which can be used as a basis for the evaluation of other projects on the PICO-8 platform.

Tabla de Contenidos

1 Introducción 1

1.1 Lua ... 2

1.2 PICO-8 ... 2

1.3 El estado de la plataforma ... 5

2 Objetivos 6

3 Metodología 7

4 Prácticas 7

4.1 Prácticas características de Lua ... 7

4.1.1 Practicas comunes ... 8

4.1.2 Artículos de Lua ... 8

4.2 Prácticas exclusivas de PICO-8 ... 11

4.2.1 Prácticas de Recursos ... 11

4.2.2 Cartridge Chaining ... 15

4.3 Corroborando las practicas ... 15

5 Estudio de Recursos 16

5.1 Shrinko8 ... 16

5.2 Metodología del Estudio ... 17

5.3 Resultados del estudio ... 18

5.3.1 Estudio de recursos ... 18

5.3.2 Datos estadísticos ... 20

5.3.3 Minificación de Comentarios. ... 25

5.4 Conclusión del experimento ... 26

6 Ejemplo de evaluación de código 27

6.1 Observaciones de la estructura del código ... 28

6.2 Utilización de Tokens ... 28

6.3 Utilización de Caracteres y Compresión ... 29

6.4 _ENV en el proyecto ... 30

6.5 Cartridge Chaining en el Proyecto ... 30

6.6 Conclusión de la evaluación ... 30

7 Conclusión y discusión 31

8 Bibliography 32

Anexo A: Resumen de practicas 35

Anexo B: Metatablas y Metametodos de Lua 37

Anexo C: Representación matemática de prácticas de la sección 4.2 38

Lista de Figuras

Fig 1: Ejemplo de modelaje en PicoCAD 3
Fig 2: Especificaciones de PICO-8 3
Fig 3: UI y editor de texto de PICO-8 4
Fig 4: Cargando proyecto en la terminal de PICO8. 4
Fig 5: Ejemplo de Utilizacion de _ENV 9
Fig 6: Demostración de Metatabla con _ENV 9
Fig 7: Demostración de funciones con _ENV 10
Fig 8: Demostración de creación de variables globales durante utilización de _ENV 10
Fig 9: Grafo de barras de tokens 19
Fig 10: Grafo de barras de caracteres. 19
Fig 11: Tabla de barras de compresión 20
Fig 12: Clusterización [Puzzles] Pre-Minificación 21
Fig 13: Clusterización [Puzzles] Post-Minificación (X v Y). 21
Fig 14: Clusterización [Puzzles] Focus Characters (X v Y). 22
Fig 15: Clusterización [Puzzles] Focus Compression (X v Y). 22
Fig 16: Clusterización [Platform] Pre y Post Minification (Tokens v Characters). 23
Fig 17: Clusterización [Platform] Todas las configuraciones (Characters vs Compression) 23
Fig 18: Porcentaje de Comentarios por Proyecto 26
Fig Anexo B: Ejemplo de Uso de Metatablas y Metametodos 37

Lista de tablas

Tabla 1 P-Values y Effect Size de las configuraciones. 24
Tabla 2 Effect Sizes del Dataset de proyectos. 25
Tabla 3 Evaluación de Utilización de Practicas de Tokens 28
Tabla 4 Minificacion Mínima de The Carpathian 29
Tabla 5 Minificacion Practica de The Carpathian 29

1 Introducción

La consola virtual PICO-8 es un artilugio interesante en cuanto a la programación de videojuegos. Además de ser una consola de videojuegos, esta contiene las herramientas necesarias para el desarrollo de juegos dentro de la misma. Sin la necesidad de utilizar hardware, además del computador en el que está corriendo, PICO-8 es capaz de simular una consola que nunca existió, con sus propias especificaciones de hardware, formato, filosofía de trabajo y una comunidad activa [1]. PICO-8 fue programado en LUA y como tal es compatible con la filosofía de desarrollo simple, portable y a pequeña escala, e incluso es capaz de utilizar archivos LUA para el desarrollo de proyectos.

En el catálogo de proyectos de PICO-8 se pueden encontrar juegos originales, 'demakes', recrear un juego en un sistema menos poderoso, y herramientas de varia índole. Un ejemplo de proyecto como ejemplo de la complejidad que se puede conseguir en la plataforma es el titulado 'Into Ruins' [2], el cual, utilizando tanto herramientas como prácticas de ahorro de recursos, fue posible de crear un proyecto con algoritmos de generación de niveles y objetos, IA para varios tipos de enemigos y un sistema original de navegación. Este proyecto realiza varias funciones complicadas dentro de los limites restrictivos, algo que se puede ver en muchos otros proyectos de la plataforma. Lo que es posible en la plataforma es solo limitado por el conocimiento del programador y su creatividad.

Cada proyecto en PICO-8 puede tener hasta 8192 tokens, o caracteres de código sin contar comentarios [3]. Esta limitación es impuesta a los desarrolladores debido a que los juegos en PICO-8 son de estilo retro, y este límite en tokens simula las restricciones de software y hardware de los motores de videojuegos que existieron al principio de la industria.

Aunque la plataforma ha encontrado un público y comunidad de desarrolladores y jugadores en el internet y existe una gran cantidad de proyectos, desde simples juegos a motores gráficos 3D, no existen estudios científicos sobre la plataforma y sus varios aspectos. Las limitaciones presentan un interesante paradigma para la mejora de prácticas de programación y la optimización del código, lo que hoy en día no se tiene en cuenta debido a cambios en prioridades de las compañías que lo desarrollan, queriendo crear un producto para satisfacer demanda, y luego si es posible, un programa estable y rápido [4].

Buenas prácticas y optimización son esenciales para desarrollar todo tipo de proyectos dentro de PICO-8, y de vez en cuando se encuentran nuevas formas de ahorrar memoria por los desarrolladores [5][6][7], pero no existe una documentación ideal o catálogo de estas buenas prácticas, además de que la validación de estas técnicas no son usualmente corroboradas, lo cual puede llevar a la mala formación de los desarrolladores más nuevos a la plataforma e incluso el uso de prácticas erróneas causando errores desastrosos en el programa.

1.1 Lua

Lua [8] es un lenguaje de programación para la creación de scripts que es ligero, poderoso, portátil y de por si fácil de aprender y utilizar. Lua como tal fue creado en 1993 como sucesor a los lenguajes DEL y SOL en la Pontifica Universidad Católica de Rio de Janeiro por el grupo Tecgraf [9]. Lua nace de la simple necesidad de incrementar las capacidades de SOL y el poder de DEL, pero decidieron reemplazar estos dos lenguajes por el titular Lua.

Este lenguaje de programación es ideal para el prototipado debido a su simplicidad y bajo tamaño, apenas llegando a 1.3MB descomprimido [8] bajo la licencia MIT. Su implementación en PICO-8 como lenguaje de desarrollo se debe a estas características además de su uso en la industria de videojuegos y su sintaxis siendo este fácil de entender [10].

1.2 PICO-8

La consola virtual se especializa en proyectos a pequeña escala, o más bien, es el propósito completo de la herramienta. Esto es, de cierta manera, forzado debido a las severas restricciones impuestas, tales como memoria, CPU y cantidad de caracteres y tokens que se pueden escribir en un archivo. Aun así, PICO-8 ha sido capaz de crear su propio séquito de fans, e incluso crear una cultura de desarrollo miniatura en los cuales aficionados desarrollan tanto juegos y proyectos como otras consolas virtuales.

Un ejemplo de proyecto realizado es PicoCAD, un programa de modelaje en 3D completamente creado en PICO-8 el cual es además capaz de exportar a otras plataformas de modelaje como Blender o motores de videojuegos como Unity [11]. También se han creado hojas de cálculo [12], sintetizadores de audio [13] he incluso se han realizado librerías de herramientas 3D [14], físicas (que es un fork de una librería ya existente para Lua) [15] y extensiones, como por ejemplo, añadir Google Analytics al proyecto [16].

Fig 1 Ejemplo de modelaje en PicoCAD

Para facilitación al lector, las especificaciones de PICO-8 que son importantes para el desarrollo de esta memoria son las siguientes:

- 32kb de información codificadas en PNG (16kb son de código).
- Código en Lua (P8 Lua) con máximo de 8192 tokens.
- Tiene un límite de caracteres de 65536.

Si existe algo que podemos optimizar o mejorar con prácticas de ingeniería se debe encontrar en estas especificaciones.

```
Display: 128x128, fixed 16 colour palette
Input:   6-button controllers
Carts:   32k data encoded as png files
Sound:   4 channel, 64 definable chip blerps
Code:    P8 Lua (max 8192 tokens of code)
CPU:     4M vm insts/sec
Sprites: Single bank of 128 8x8 sprites (+128 shared)
Map:     128 x 32 Tilemap (+ 128 x 32 shared)
```

Fig 2: Especificaciones de PICO-8 [2]

En cuanto a su relación con Lua, PICO-8 utiliza algo cercano al lenguaje de programación con sus diferencias únicas, como ciertos atajos a algunas funciones y operaciones. Existe un git fork de Lua que integra estas características y es compatible por mayor parte a PICO-8 llamada z8lua [17]. Un Token en este caso se trata de cualquier elemento en el código que tenga un valor significativo para su funcionamiento, como una variable, función u operación.

La resolución mostrada en las especificaciones no solo se refiere al tamaño de la ventana en la que se correrá el proyecto, sino al de la plataforma en sí, lo cual reduce el tamaño que tenemos para escribir código y la cantidad de elementos que

podemos visualizar al mismo tiempo, algunos de estos siendo las herramientas de dibujo, mapeo y efectos de sonido. Aun así, el código se puede dividir en partes gracias a las pestañas en las que podemos separar el proyecto según cualquier criterio que el programador crea conveniente.

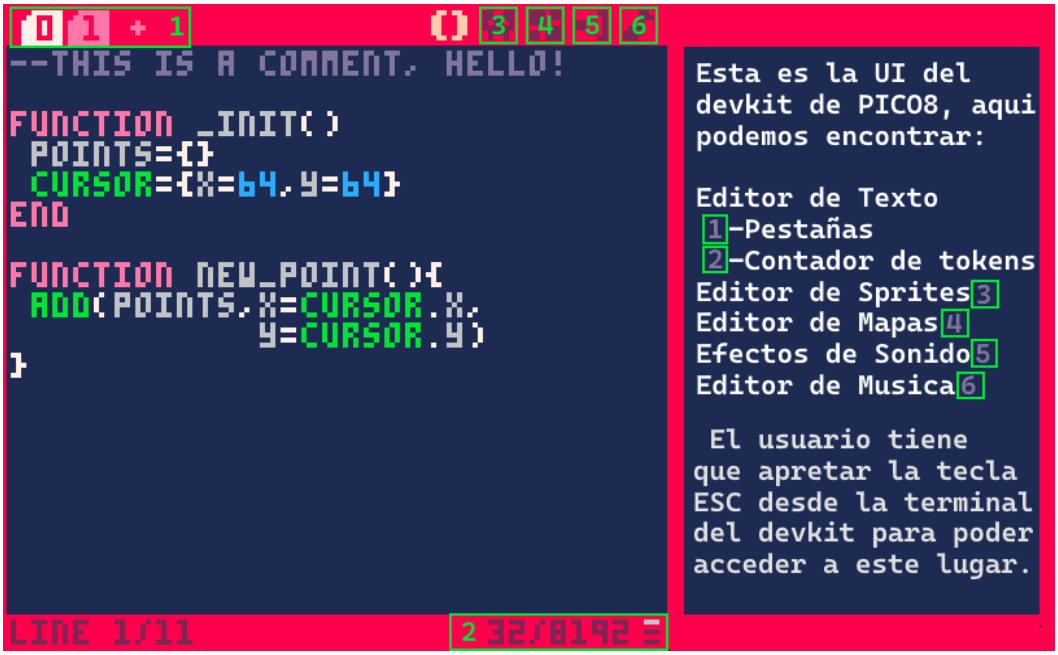

Fig 2 UI y editor de texto de PICO-8

El formato en el que se guardan los proyectos es “.p8” el cual en sí es solo un texto con el código y otros datos en hexadecimal, pero también puede ser codificado en un PNG que cumple el mismo propósito, pero con algo de presentación.

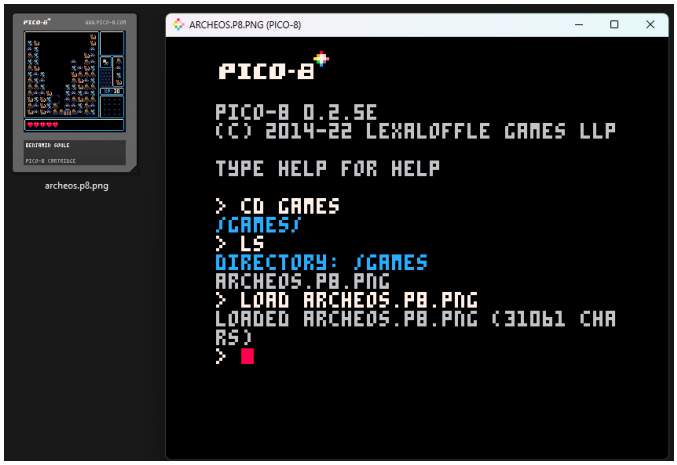

Fig 3 Cargando proyecto en la terminal de PICO8.

Por último, es importante definir cada recurso que se busca optimizar o necesidad que se busca lograr al escribir un proyecto en PICO-8:

Tokens [18]: Son elementos que representan algún valor, operación, variable, palabra clave o encapsulación dentro del código, sin contar los elementos que sean delimitación como la palabra clave ‘End’ o ‘;’. Los tokens son la representación más importante de las limitaciones de la plataforma ya que cada una representa una acción, objeto u valor que sean necesarios para la ejecución del código. Soluciones creativas o buenas prácticas de código son necesarias para aprovechar la mayor cantidad de tokens posible.

Caracteres: Estos elementos representan los límites de la memoria del cartucho que existían en décadas posteriores. Cada carácter es contado para este límite, incluso los comentarios de cualquier tamaño lo cual puede resultar molesto cuando el programador requiere de una documentación concisa de lo que hace su código si este fuese un proyecto de varias personas. La utilización de este recurso es dependiente del tamaño del proyecto; mientras más cercano al límite, menor cantidad de comentarios, e incluso número de caracteres por variable se pueden utilizar.

Compresión [19]: Este es el recurso que menos importancia se le da dependiendo de las necesidades del proyecto. La “Compresión” de un proyecto se refiere al tamaño del mismo cuando este se guarda en el formato ‘.p8.png’ como se muestra en la figura 1.3. Durante el desarrollo, el programador puede ver el porcentaje del tamaño total utilizado, comparado con los otros dos recursos que muestran las unidades y su máximo. Este recurso es *completamente opcional*, ya que si el proyecto es guardado en el formato ‘.p8’, este no contará como parte de las limitaciones. La limitación es para crear un cartucho digital de pequeño tamaño que pueda ser presentado y estéticamente similar a otros proyectos, dando un sentido de identidad a los juegos y herramientas desarrollados en la plataforma.

El balance de estos tres recursos es necesario para máxima eficiencia de desarrollo, pero puede resultar complicado para un programador que no tenga una disciplina de código limpio y aun así los límites pueden ser muy estrechos para el desarrollador promedio. Aun así, estas limitaciones no deben verse como problemas a eliminar, sino como puzles donde el ingenio y creatividad son necesarios para resolverlos. Para ello, se han recopilado ciertos tips y prácticas de programación las cuales pueden ayudar al desarrollador para realizar estos puzles en el capítulo 4.

1.3 El estado de la plataforma

Actualmente PICO-8 cuenta con una comunidad vibrante y amigable la cual discute sobre las nuevas versiones, juegos; es popular, pero no es grande. Existe una wiki que contiene varias características del juego que sigue siendo actualizada

[20]. Además de esto, los foros del desarrollador son activos todos los días, con nuevos proyectos y noticias [21].

Aun así, no existe algo como un recurso que les permita a los usuarios ver consejos de programación o los que existen son descentralizados, y, aunque existen varios recursos sobre buenas prácticas, pocos hay de malas prácticas. Aun así, la mayoría de estas malas prácticas pueden considerarse como el opuestos de estas buenas.

El propósito de este documento es la identificación de prácticas existentes descritas por la comunidad de PICO-8, analizarlas de forma científica y mencionar sus casos de uso, ventajas y desventajas, con el propósito de optimización de los recursos de la plataforma, el análisis de herramientas, como los recursos interactúan entre sí, y por último tomar un proyecto para analizarlo utilizando lo estudiado.

2 Objetivos

Para realizar esta tarea, tenemos como objetivo general la recopilación de las practicas que se han ideado por la comunidad, describirlas y analizarlas, junto con sus casos de uso. Luego buscar herramientas que puedan ayudar en el desarrollo y escritura de código para entonces realizar pruebas con las mismas. Al final evaluaremos el código de un proyecto utilizando dichas prácticas y herramientas y como estas pueden mejorar varios aspectos de este.

Utilizando nuestro objetivo general, se tienen varias tareas a realizar que se pueden dividir de la siguiente manera:

1. Definir lo que es una práctica y dividirlas en varias categorías según su origen, propiedades y como actúa en la plataforma PICO-8.
2. Describir el propósito de las practicas, por qué utilizarlas y recalcar los casos en que estas se pueden considerar malas prácticas.
3. Describir las herramientas desarrolladas para la plataforma que pueden realizar labores de desarrollo en el transcurso de un proyecto.
4. Realizar pruebas y análisis de datos sobre los recursos utilizados en un dataset de proyectos y el posible uso de las herramientas descritas.
5. Tomar un proyecto y realizar su análisis; ver cómo podrían afectar las practicas a los recursos y la estructura del código con el propósito de verificar su necesidad en el proyecto.

3 Metodología

Durante la investigación de esta memoria, no se encontraron documentos científicos en los cuales PICO-8 sea el protagonista, usualmente es mencionado por otros aspectos que no son los que se presentan en este documento. La mayoría de nuestras fuentes resultan ser de miembros de la comunidad en los foros, proyectos de GitHub y páginas personales de programadores, aunque varias de estas se corroboran entre sí.

Para la experimentación, se tomará una herramienta en particular que nos permitirá la optimización de recursos y se pasaran por un dataset de proyectos, con el cual visualizaremos las relaciones entre recursos utilizando distintas tablas comparativas y cómo podemos utilizar esta herramienta durante el desarrollo de un proyecto

Por último, la versión de PICO8 utilizada es la v0.2.6b, la más actual al momento de escritura; la lista de versiones se puede encontrar en la siguiente referencia [22]. Además, se utilizó Python 3.10 junto con un notebook jupyter para la recolección de datos en el estudio.

4 Prácticas

Para efecto de la memoria se referirá como prácticas a técnicas y nociones de programación que son ideales para el ambiente de un proyecto y que permiten al programador optimizar el código ya sea de forma técnica (Uso de CPU o cantidad de código) o legibilidad para otros programadores. Estas prácticas serán divididas según si están relacionadas a las varias características de Lua como lenguaje o si son sintácticas o propias de PICO8 como plataforma.

En el anexo A se encontrará una tabla la cual se puede usar como referencia y resumen de las practicas a hablar en este documento. Estas aparecerán en orden según aparezcan en el informe y presentan los siguientes datos en inglés: Nombre, Resumen, Ventajas y Fuentes.

4.1 Prácticas características de Lua

Podemos dividir estas prácticas en prácticas comunes, que tienen relación con prácticas generales de programación que tiene son válidas al programar con Lua, y prácticas que utilizan artículos de Lua, elementos específicos que nos permitirán la optimización de recursos y ordenar código.

4.1.1 Practicas comunes

Como primera referencia se utilizará una recopilación de “Best Practices” para Lua [23]. Aquí hay muchas prácticas que se pueden considerar comunes u obvias, pero se destacarán algunas, teniendo en consideración las limitaciones impuestas por PICO8:

- Nombres Cortos y Acrónimos: Estas prácticas son mostradas como code smell (mala práctica de código). El hecho de utilizar abreviaciones para nombres de variables y funciones puede resultar en una experiencia incómoda para el lector. Un buen nombre resume la funcionalidad de una función y el de una variable para mejor código en general, pero en PICO8 no tenemos ese lujo. Mientras más corto el nombre mejor, pero aun así tiene que ser entendible de qué va el nombre.

- Límite de caracteres por línea: El documento menciona que 80 caracteres por línea es lo ideal para mejorar lectura, lo cual es una proposición lógica. Pero la plataforma PICO8 tiene una resolución de 128x128 pixeles, el tamaño de los caracteres fuente no se pueden cambiar, y antes de que la pantalla se corre horizontalmente para acomodar el texto, se pueden escribir 32 caracteres. Lo ideal es minimizar la cantidad de caracteres manteniendo coherencia con el código, por esto mismo se propone un límite de 28 caracteres para dejar una sangría.

- Comentarios: Se tienen que mantener al mínimo ya que los comentarios son contados para el límite de caracteres. Si son absolutamente necesarios, lo ideal es exportar el proyecto a alguna plataforma para compartir proyectos (GitHub) y realizar los comentarios ahí. De otra forma, se deberá tratar de escribir los comentarios de manera concisa.

4.1.2 Artículos de Lua

Lua está estructurado de tal forma que contiene varias funciones y características que las definen como lenguaje, hay una en especial que nos será muy útil la cual utilizará los artículos de Lua denominados **metatablas y metametodos** [24] (Anexo B).

La razón por la que queremos utilizar estos artículos es debido a que Lua nos permite cambiar en “ambiente” de trabajo a una tabla, en otras palabras, las nuevas variables creadas serán parte de la tabla sin tener que mencionarla en el código.

_ENV [26] es un artículo de LUA que heredó PICO-8 que puede ser utilizado para la optimización de tokens. Todas las variables dentro de un código pertenecen al _ENV, pero también podemos asignar una tabla que sea nuestro _ENV de forma local. Así, cada vez que utilizamos una variable de la tabla o creamos una nueva, se interpretará sin la necesidad de declarar el uso de la tabla misma:

```
player={}
player.x=10
player.y=20
player.lives=3
player.sp=1
```

```
player={}

do
    local _ENV = player
    x=10
    y=20
    lives=3
    sp=1
end
```

Fig 4 Ejemplo de Utilizacion de _ENV

Como requerimiento, es necesario utilizar _ENV de forma cursiva en PICO-8 [42], tal que sea escrito de la forma *_env.*

Una gran desventaja es el hecho que sobrescribe el ambiente global, por tanto, todas las definiciones, variables, caracteres especiales, entre otras definiciones de PICO8 globales no son accesibles mientras se esté en el _ENV local. Para solucionar este problema es necesario utilizar metatablas junto con el metametodo __index. Este último tiene como función buscar el elemento al que pertenezca una llave la cual no se encuentre en la tabla. Por tanto, si utilizamos metatablas para integrar el metametodo __index a una tabla, podemos utilizar características y variables globales sin mayor inconveniente.

Aquí no se debería imprimir 'd', pero debido a la metatabla, __index buscará en el ambiente global la variable 'd' y lo imprimirá correctamente.

```
d=4
mytable={
    a=1,
    b=2,
    c=3,
}
setmetatable(mytable,{__index=_ENV})
?mytable.a --Prints 1
?mytable.b --Prints 2
?mytable.c --Prints 3
?mytable.d --Prints 4
```

Fig 6 Demostración de Metatabla con _ENV

También es útil utilizarlo dentro de funciones:

```
tbl=setmetatable({a=1},{__index=_ e n v })
b=2
function doit(_ e n v )
 ?"tbl.a=>"..a
 ?"global b=>"..b
end
doit(tbl)
```

Fig 7 Demostración de funciones con _ENV

Ahora tenemos que la función no debería correr debido a que 'a' no se encuentra como 'tbl.a' pero debido a la metatabla, es parte del ambiente global y es capaz de ser llamado gracias a __index.

El único problema que surge es no poder crear nuevas variables globales mientras se esté en el ambiente local creado. La solución para este caso sería crear una variable que apunte al ambiente global tal como 'global = __env' antes de interactuar con el ambiente local. Lua de por si viene con dicha variable llamada '_G' pero en el caso de PICO-8 se tendrá que crear una que lo contenga.

```
-- used by __index access
_g=_env

mytable=setmetatable({
a=0,b=0,c=0,
init=function(_env)
 a,b,c=1,2,3
 newvar=25 -- this is created in mytable
 _g["initialized"]=true -- this is created in GLOBAL ENVIRONMENT
end
},{__index=_env})

mytable:init()

function _draw()
  -- outside table scopes (no __index access)
  local _g,_env=_env,mytable
  if _g.initialized and _g.btnp()>0 then
  a+=1
  b+=1
  c+=1
end

cls()
?tostr(initialized)..":"..mytable.newvar.., "..mytable.a..", "..mytable.b..", "..mytable.c
end
```

Fig 8 Demostración de creación de variables globales durante utilización de _ENV

4.2 Prácticas exclusivas de PICO-8

Estas prácticas se diferencian con las anteriores por el hecho de que no están sujetas a funciones o elementos de Lua, sino a la sintaxis presente en PICO-8 y sus elementos o técnicas más específicas, buscando aumentar la cantidad de recursos disponibles para el proyecto.

4.2.1 Prácticas de Recursos

Aquí se busca tomar partes de código y reemplazarlas por código equivalente que sea menos costoso en recursos. Se dividirán las practicas según el recurso que influencien.

Tokens [5]**:**

1. Default arguments
    - Cuando se utiliza una función, si existen varias con el mismo propósito, es ideal tratar de utilizar las funciones con menor cantidad de argumentos. Un ejemplo es btn() y btnp(); la diferencia siendo que btnp() requiere el ID del jugador, añadiendo un token extra.
    - Cuando se crea una función, sería ideal que, si la función sea llamada varias veces con el mismo valor de argumento, decirle a la función que su argumento sea igual a dicho valor si esta estuviese vacía:

```
function foo(argument)
    --do something
end

foo(5)
foo(5)
foo(5)
foo(5)
foo(5)
foo(5)
```

```
function foo(argument)
      argument=argument or 5
      --do something
end
foo()
foo()
foo()
foo()
foo()
foo()
```

En este ejemplo, el primer programa utiliza 22 tokens, mientras que el segundo utiliza 21.

2. Asignar con comas
    - Esto refiere a tratar de asignar valor a la mayor cantidad de variables posibles en una sola sentencia para reducir la cantidad de = utilizadas dentro del código de la forma:

```
var1, var2 = value1, value2
```

    - Un problema encontrado dentro de esta práctica es la asignación de dos valores en el cual uno toma el valor antiguo del otro:

```
x,y=1,2; x,y=3,x
```

En este caso X toma el valor de 3 mientras que Y toma el valor de 1, el valor antiguo de X.

- También existe el caso de asignar valor Nil a una variable utilizando esta práctica de tal modo que sea de la forma:

```
var1,var2 = value1
```

3. Operaciones matemáticas
   - Utilizar operaciones matemáticas entre constantes puede resultar en mayores cantidades de tokens utilizados. Es ideal calcular el resultado de la operación y escribirlo en el código.
4. Inicialización inicial de Tablas de Lua
   - Esta es una regla general; Utilizar tablas constantemente genera más tokens que cualquier otra acción dentro del código. Se puede considerar una mala práctica en ciertos casos.
   - Es ideal utilizar variables locales dentro de funciones y darles los argumentos relevantes de la tabla en vez de utilizar la tabla completa.
   - Si hay que crear una tabla, es ideal inicializar todas las propiedades requeridas al momento de declarar la tabla, ahorrando 1 token por cada propiedad declarada.
5. Array Indexing
   - Preferir usar llamadas a propiedades de una tabla en vez de llamar una variable usando un array, ahorrando 1 token.
   - En este mismo caso, si fuésemos a utilizar vectores, podemos crearlos de dos maneras:

```
--Declaration-------Access---------------
  point={x,y}     / point[1] and point[2]
  point={x=x,y=y} / point.x and point.y
```

   El primer caso requiere 5 tokens para crear, pero 3 para acceder a los datos, mientras que el segundo requiere 9 para crear y 2 para acceso.
6. Logical Short Circuiting
   - Este es un aspecto más profundo de la jerarquía de programación encontrada en LUA en el cual en “foo() or bar()” nunca correrá bar() si foo() es ‘verdadero’. Esto se puede utilizar para dar valor a una variable dependiendo del resultado de varias funciones de la forma:

```
thing = foo() or bar() --Is equivalent to:
thing = foo() if(not thing) thing bar()
```

   - Esto se puede utilizar en operaciones aritméticas tal que si una operación requiere de foo() para actuar, podemos reducir el código: por ejemplo:

```
if foo() then a = b+c else a = b+d end --Can be:
a = foo() and b+c or b+d --Reduced 3 tokens
```

7. Calling Functions with strings or tables
    - Se puede llamar a una función de varias maneras en el caso que el argumento sea una tabla o string de manera que podemos ahorrar un token:

```
func("STRING") and func({TABLE}) --3 tokens
func"STRING" and func{TABLE}     --2 tokens
```

    - Esto solo funciona cuando la función utiliza un solo argumento.

8. Parsing data from String literals.
    - Al almacenar grandes cantidades de información en una tabla ya sea para diálogo, coordenadas u otro tipo de cosas, se puede fácilmente llegar al límite de tokens. Se puede almacenar esta información utilizando strings y realizando parseo:

```
data_string="0,1,2,3,4,5,6,7,8,9,8,7,6,5,4,3,2,1,2,3,4,5,6,7,8,9"
data=split(data_string)
for i in all(data) do
    print(i)
end
```

    - Un problema que puede surgir de esto es la utilización de otros recursos como la memoria. Lo ideal sería utilizar este método cuando quedan recursos de sobra mientras que se está cerca del límite de tokens.

Caracteres [27]**:** Tweetcarts, una sub-rama de desarrollo que se caracteriza por un límite de 280 caracteres, nos pueden enseñar varios métodos de optimización de caracteres con el propósito de realizar mayores proyectos sin tocar el límite de caracteres.

9. Line breaks
    - En algunos casos es posible eliminar saltos de línea con el propósito de disminuir caracteres, concatenando varios tokens a la vez, los cuales pueden ser funciones o variables:

```
cls()  to cls()p=5
p=5
```

    - Si queremos concatenar líneas, lo ideal sería que terminaran en funciones tal que terminen con paréntesis u otros caracteres:

```
x=y+sin(a)b=x
```

    - También podemos concatenar varias variables a las cuales se les asigna un valor:

```
a=5z=10r=0
```

      Pero hay que tener precaución de utilizar las letras x o b, ya que 0x y 0b son definiciones para números hexadecimales y binarios.

10. Function renaming
    - Podemos almacenar el nombre de una función en una variable y utilizarla como si fuese la función misma:

```
s=sin
p=print
p(s(0))
```

    - La función print se puede simplificar aún más si solo se utiliza un '?' y sin paréntesis, incluso es capaz de aceptar múltiples argumentos:

```
?"hello, tailor",64,64,time()*8
```

11. If-statements
    - Simplemente eliminar las sangrías usuales de condicionales y, en Lua, "Then" y "End" pueden estar junto a cualquier tipo de declaración, como si fuesen el final de un paréntesis. Además de esto, If statements in Lua pueden no estar entre paréntesis.

```
if t()>5then
print("hello, tailor")end
```

    - También es posible eliminar la necesidad de utilizar "Then" y "End" si el condicional es lo suficientemente pequeño:

```
if(t()>5)print("hello, tailor")
```

      Esto se puede combinar con la práctica del punto 9b para concatenar la mayor cantidad de funciones posibles.

```
if(t()>5)print("hello, tailor")print("and goodbye")
```

Hay que tener cuidado al respecto de estas últimas prácticas debido a lo destructivas que pueden ser en el proyecto. Se refiere a la eliminación de legibilidad del código ya que lo que se recomienda es eliminar nuevas líneas, comentarios y reducción del nombre de variable y funciones lo más posible.

Estas prácticas son esenciales si se tiene un límite extremo de caracteres donde poner el código, mientras que en proyectos más grandes esto resulta en prácticas que solo los proyectos más complejos deberían utilizar si se encuentran con estos límites después de haber realizado buenas prácticas de código durante todo el proyecto.

Compresión: No hay una forma fácil de cuantificar el uso de la comprensión del código, y usualmente no suele ser un problema *a menos* que haya una utilización extensiva de strings y comentarios que sean largos [28] debido a la entropía del código. En este caso llamamos entropía a la necesidad de incrementar la cantidad de bits utilizados para el almacenamiento de los datos al momento de compresión debido a la cantidad única de datos que pueden existir dentro del código [29]. Se sugiere en este caso utilizar las prácticas anteriormente mencionadas en la sección 4.1 para la disminución del tamaño de la compresión en el proyecto.

4.2.2 Cartridge Chaining

Aunque se halla planeado como utilizar cada recurso, no es difícil llegar a sus límites. Feature Creep [30] es un problema al que se puede llegar desarrollando proyectos de programación, el cual impacta fuertemente a PICO-8 debido a estos límites, pero si utilizamos varios “proyectos” a la vez, podemos ser capaces de realizar todo lo necesario e ignorar los límites de la plataforma.

Utilizando dos o más cartuchos (proyectos de formato ‘.p8’ o ‘.p8.png’), podemos aprovecharnos de nuestros nuevos límites extra para realizar distintos tipos de cálculos, funciones y almacenamiento de información para luego pasarlos directamente a la RAM, cargar el cartucho principal, y utilizar o almacenar la información obtenida [31]. Se puede decir que esto hace referencia a los días en que los juegos venían en más de un disco, y cuando llegaba la hora de pasar al siguiente, se cargaba a la RAM la información necesaria para autorizar la continuación al próximo disco. Este sería el único método que nos permite pasarnos de los límites de la plataforma siguiendo en ella que aun así sigue siendo un caso extremo ya que puede surgir la gran excusa de que al tener más recursos uno perfectamente puede ampliar la escala del proyecto.

Ciertos proyectos, normalmente juegos, utilizan Cartridge Chaining para proyectos que requieren de amplias cantidades de datos o niveles que, en código, pueden resultar tan grandes como varios proyectos. Para ejemplo de esta técnica está POOM [32][33], el cual es un demake del juego DOOM, que contiene más de 20 archivos ‘.p8’ hasta la fecha.

4.3 Corroborando las practicas

Las practicas anteriormente mencionadas fueron recolectadas de solo algunas fuentes las cuales miembros de la comunidad corroboran su utilización como consejos de programación para el desarrollo de proyectos e incluso para herramientas externas.

La discusión de prácticas comunes de programación no es variada; un usuario puede elegir entre utilizar la interfaz de PICO-8 o utilizar un editor de texto siguiendo las reglas de compilación de cartuchos de PICO-8 para evadir utilizar la ventana de la plataforma. Por lo mismo es posible trabajar en un repositorio como Github con varias personas sin interactuar con el programa, lo cual inutiliza las practicas reconsideradas en la sección 4.1.1, pero esto no las invalida. En sí, entonces, lo importante es tener en cuenta el formato de trabajo de los usuarios al momento de aplicar las practicas.

PICO-8-Token-Optimizations [5] es la fuente más mencionada cuando se habla del sujeto. Se puede ver, e incluso es mencionado, la utilización de estas prácticas en juegos como ‘Into Ruins’ [2], demos avanzados que demuestran varias capacidades de PICO-8 como PICOCHAK [34], incluso utilizando Jinja2 para crear un preprocesador que simule la funcionalidad de #def en el lenguaje C para mejorar legibilidad pre-procesamiento y optimizar caracteres post-procesamiento [35].

En el sitio web de la compañía desarrolladora Lexaloffle, se tiene un foro donde usuarios discuten sobre proyectos y otros tópicos relacionados a los varios productos de la empresa. Una búsqueda rápida en los foros nos muestra siete publicaciones que mencionan al recurso; buscando en Google, obligando a la búsqueda a contener las palabras “PICO-8 Token Optimizations”, nos da 88 resultados que varían entre recomendaciones y citaciones. Podemos corroborar que, si, las practicas son estudiadas y aplicadas, pero en una escala no demostrable debido a la baja cantidad de menciones, aunque estas sean un tema común de discusión en la comunidad. Debido a esto, se realizaron demostraciones matemáticas de ciertas prácticas. De estas, solo 3 requieren demostración debido a que su utilización depende de factores lógicos, las prácticas de la sección 4.2; 1, 5 y 6; Default Arguments, Array Indexing y Logical Short Circuiting. (Anexo C) .

5 Estudio de Recursos

En esta sección se hará un estudio con el propósito de observar la utilización de recursos, datos estadísticos, la relación entre estos en varios proyectos, y en los varios factores que pueden ser optimizados utilizando la herramienta Shrinko8 para integrarla dentro de un flujo de trabajo donde esta pueda reducir la utilización de recursos cuando se requiera.

5.1 Shrinko8

A través de los años se han creado varias herramientas para la plataforma, desde plugins, exportadores e importadores, hasta programas externos que nos permiten el preprocesamiento de los cartuchos para varios propósitos. La herramienta Shrinko8 [37] tiene como propósito la minificación (reducción) del tamaño de código de tal manera que se reduzcan los tokens, caracteres y compresión del proyecto. Es una herramienta completa con varias funciones, configuraciones y usos extra como linting y exportación de formatos ‘.p8’ a ‘.p8.png’ y viceversa que funcionan apropiadamente. Si hay algo que criticar, sería que la complejidad del programa, en cómo está estructurado, requiere de una documentación más en detalle que lo provisto actualmente en cuanto al contenido del código.

Para utilizar la herramienta, uno primero debe tener un cartucho que analizar en un directorio, darle al programa el directorio del output y las configuraciones deseadas, estas configuraciones nos pueden dar un código similar al del input o caóticamente diferente, pero con la misma funcionalidad. Cabe destacar que la herramienta es destructiva y el output no puede ser considerado un reemplazo para el código original.

5.2 Metodología del Estudio

- Se utilizará un dataset llamado PICOwsome [42] de un total de 2966 proyectos creados por la comunidad completamente funcionales, separados en 18 categorías según su género, lo que facilitará el estudio del proyecto común y su utilización de recursos. La única presentación limitada con este set es la falta de juegos con Cartridge Chaining debido a que estos no son soportados por archivos '.p8.png' y el límite de compresión debe ser estudiado.
- Usando Shrinko8 se hara lo siguiente:
    - La configuración '--count' nos permite ver el uso de recursos por el proyecto según número y porcentaje utilizado. Utilizaremos porcentaje para tener datos normalizados.
    - Los proyectos se minificarán utilizando 4 tipos de configuración: '--minify', '--focus-tokens', 'focus-chars', '--focus-compressed'.
    - Junto con los datos de input, tendremos 5 sets de datos para revisar utilizando la configuración '--count'.
    - Pruebas pequeñas no relacionadas a los pasos anteriores que nos pueden servir para analizar problemas comunes en el código de los proyectos.
- Compararemos los datos según sus recursos en un antes y un después, el coeficiente relacional entre recursos utilizando las categorías para tener una vista más individual de los datos, junto con el P-Value y Effect Size de cada recurso.
    - El coeficiente de correlación de Pearson se puede definir como un índice que puede utilizarse para medir el grado de relación de dos variables siempre y cuando ambas sean cuantitativas y continuas el cual se interpreta en un set de [-1,1]; si es positivo, significa que, si una variable aumenta, la otra también lo hará, si es negativo lo inverso ocurre, según el valor del coeficiente. Si es 1 o -1 significa que es perfectamente positiva o negativa, si es 0 significa que no existe relación alguna entre los datos.
    - P-Value es el valor probabilístico de que una hipótesis nula sea verdadera, para lo cual debe tener un valor mayor a 0.05, y

nosotros buscamos lo opuesto, un valor menor a 0.05 [38] para luego calcular el Effect Size.
    - El Effect Size es la escala con la cual se puede apreciar el efecto de alguna función, herramienta o cambios que ocurrieron en los datos por cualquier motivo, en nuestro caso, la minificación de los datos.
- Se hará discusión de los datos obtenidos y cómo reflejan con las prácticas de programación.

5.3 Resultados del estudio

Se separarán los resultados en la utilización de recursos del dataset, como se relacionan estos entre si y como afectan los comentarios a los proyectos.

5.3.1 Estudio de recursos

En esta prueba pondremos tres configuraciones juntas; Pre-minificación, Post-Minificación y Focus ‘c’, donde ‘c’ es el tipo de recurso a comparar, además, los datos se separarán en percentiles de 5%. Los siguientes grafos tienen como eje ‘x’ los percentiles de los recursos utilizados (%R) y el eje ‘y’ el número de proyectos por percentil (Pr).

En el primer caso, visualizando tokens, podemos ver los datos repartidos en todos los percentiles casi de forma equitativa salvo en el primer y último percentil. El percentil de 95% presenta la mayor cantidad de datos, alrededor de un 4% del total; muchos proyectos en la plataforma llegan a este límite debido a su complejidad o la mala utilización de recursos, aun así, este dato no es significativo por sí mismo. En la minificación de los tokens, el cambio más observable es el grupo mencionado anteriormente. Con una minificación normal, 50 de los proyectos bajaron del percentil y con la tercera configuración bajaron otros 50 más. Por ahora no podemos hacer conclusiones de la calidad del código, pero lo que si podemos observar es que alrededor de un 20% de los datos en el último percentil muestran capacidades de mejora en cuanto a la optimización de tokens.

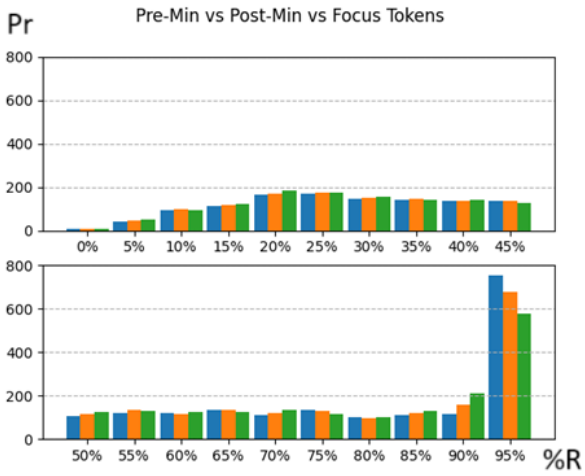

Fig 9: Grafo de barras de tokens

En el siguiente caso tenemos los caracteres los cuales, como dicho anteriormente, representan un límite más amplio, pero todo carácter cuenta. Aunque, a lo menos para los proyectos de este dataset, una cantidad insignificante llegó a los percentiles finales, pero su minificación es superior en creces comparada con los tokens. Podemos observar cómo los proyectos de los percentiles altos son casi inexistentes después de la minificación, la gran mayoría en los percentiles sub-50. Hay que recordar que las prácticas de reducción de caracteres son altamente destructivas, reduciendo el código a solo un par de líneas en algunos casos.

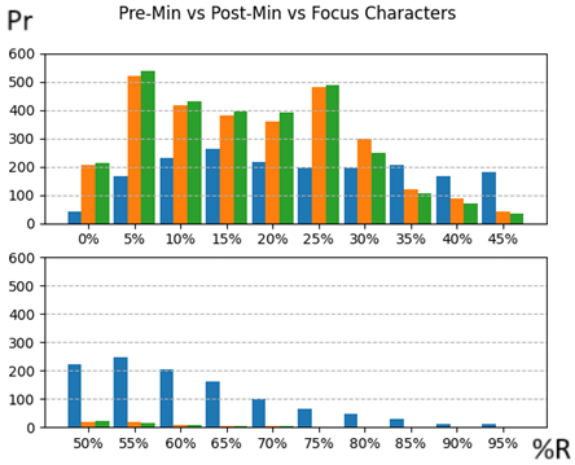

Fig 10: Grafo de barras de caracteres.

Por último, tenemos la compresión del código la cual esperamos tener un mayor entendimiento sobre su funcionamiento durante el estudio. En este caso, los resultados pueden verse relativamente equitativos comparado con los recursos anteriores, incluso después de la minificación.

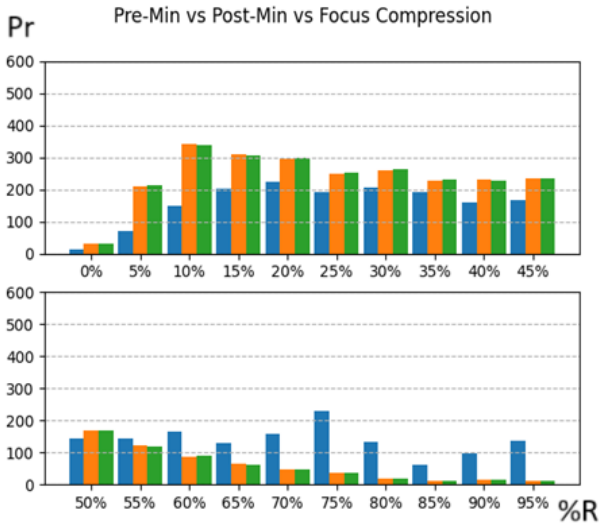

Fig 11: Grafo de barras de compresión

5.3.2 Datos estadísticos

En el siguiente experimento se buscará el coeficiente relacional, con el que se espera ver algún tipo de relación entre la minificación Focus de los recursos los cuales se representarán en un plano de coordenadas, similar a la clusterización de datos. Para esto se va a reducir la muestra a dos categorías para mayor legibilidad de los datos. Elegiremos el segundo y el cuarto más poblado que son '[Puzzle]' y '[Platform]'. En la primera categoría, haremos un análisis completo y realizaremos una hipótesis, mientras que en la segunda veremos los datos que la corroboren o nieguen.

Como primera observación pre-minificación tenemos que, en todos los casos, los datos se conglomeran en los bajos percentiles y de a poco se empieza a ampliar la varianza de recursos utilizados por proyecto. Los coeficientes de relación en este, como en todos los casos, son muy altos, en especial caracteres y compresión. Cabe destacar la similitud entre los grafos que comparan tokens con otro recurso; ambos presentan una forma similar y coeficientes cercanos, además, cuando los tokens se ven cerca del máximo, la varianza del otro recurso llega a su mayor punto.

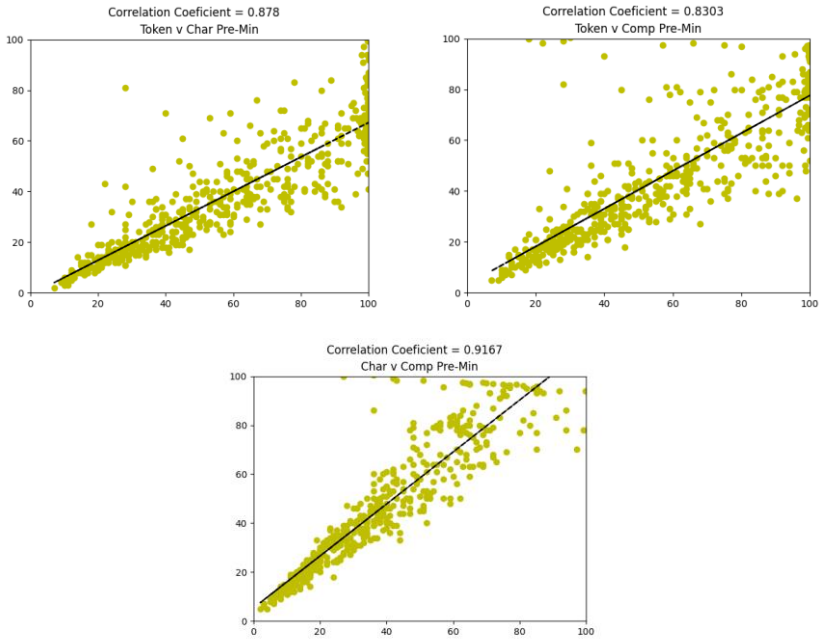

Fig 12: Clusterización [Puzzles] Pre-Minificación

Post-minificación, los coeficientes relacionales de la clusterización que comparan tokens son menores que los anteriores mientras que el último incrementa. Con esta prueba, junto con la del experimento anterior, podemos corroborar la inefectividad de la herramienta en la minificación de tokens comparado con los otros recursos, incluso podemos formular otra interrogante, ¿Qué tan dependiente son los recursos de los tokens?

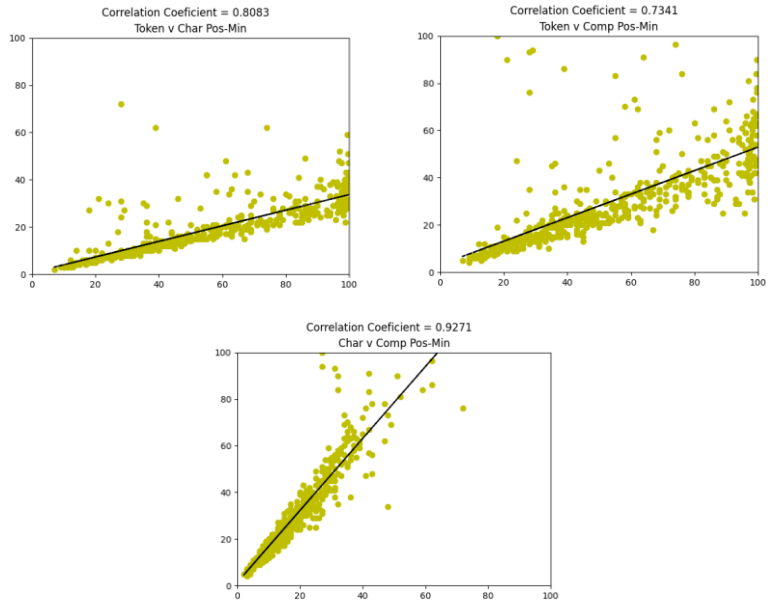

Fig 13: Clusterización [Puzzles] Post-Minificación (X v Y).

Realizando Focus Tokens no realiza cambios significativos en la utilización de recursos y el coeficiente de correlación es similar al caso anterior. Para el resto

del experimento, no mostraremos la clusterización de recursos donde haya Focus Tokens.

En nuestras dos últimas configuraciones, la mayor diferencia se presenta en, irónicamente, la comparación de tokens con otros recursos, aunque es pequeña. En las comparaciones de caracteres y compresión, podemos apreciar que el coeficiente de correlación es casi idéntico. Podemos hipotetizar que focus caracteres y compresión tienen aproximadamente el mismo efecto en el código, pero no sabemos a qué se puede deber esto. Sería ideal en este caso realizar una prueba menor en los distintos elementos que pueden ser modificados por separado y ver si alguno de estos tiene un efecto equivalente tanto para caracteres y compresión.

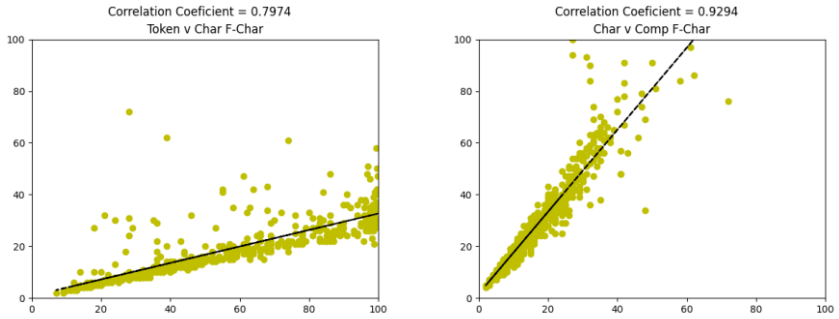

Fig 14: Clusterización [Puzzles] Focus Characters (X v Y).

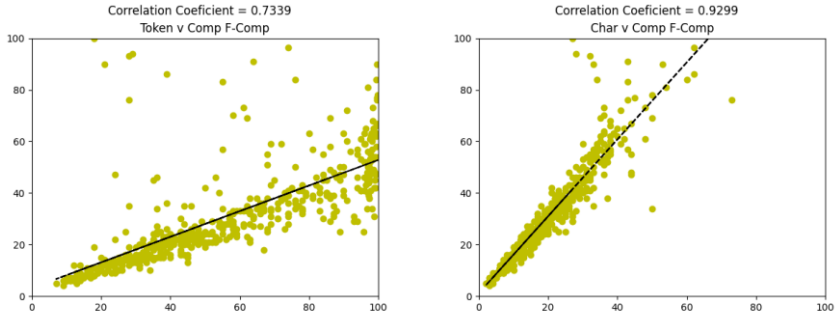

Fig 15: Clusterización [Puzzles] Focus Compression (X v Y).

Ahora se verá la categoría [Platform] para contrastar con los datos anteriores. Se compararán tokens versus caracteres pre y post, y luego solo veremos todas las clusterizaciones de caracteres versus compresión si la situación de los tokens es similar al caso anterior.

Como se puede observar, el coeficiente relacional **baja** después de la minificación como en el caso anterior, incluso los números son similares. La similitud de estos resultados tomando en cuenta que ambos son categorías separadas con distintos propósitos, podemos concluir que tratar de observar el cambio de tokens

no nos da información relevante comparando con los otros dos recursos utilizando esta herramienta.

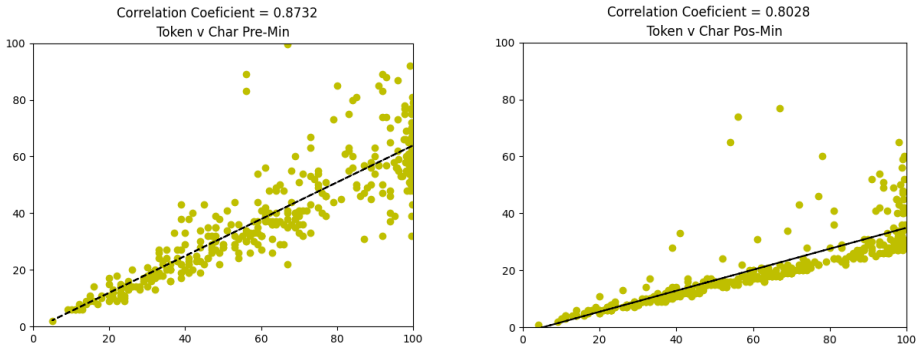

Fig 16: Clusterización [Platform] Pre y Post Minification (Tokens v Characters).

El aumento del coeficiente de correlación es mínimo, pero suficiente para ver que el uso de caracteres y compresión están altamente relacionados entre sí. El hecho de que el coeficiente más alto sea con focus compresión puede significar que la disminución de caracteres es más agresiva dentro de la configuración.

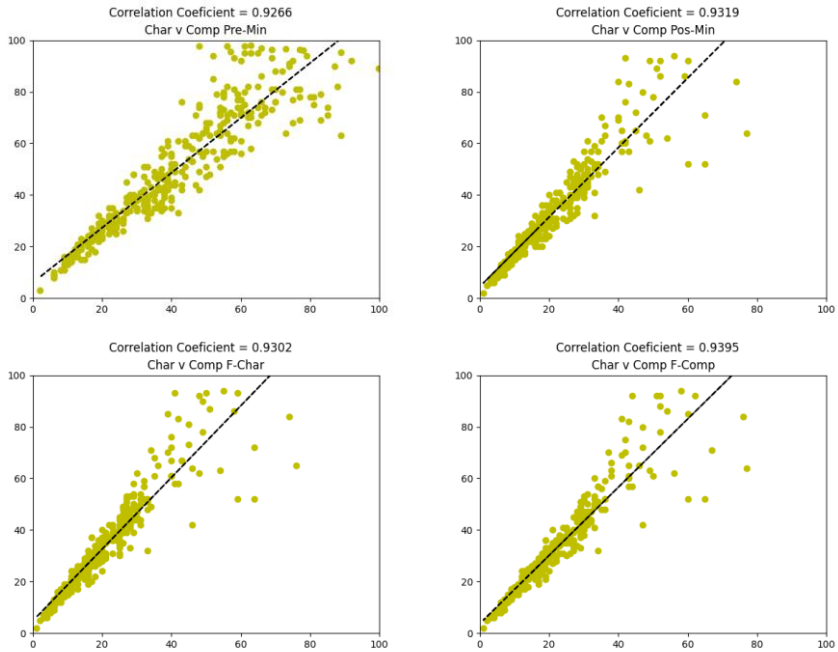

Fig 17: Clusterización [Platform] Todas las configuraciones (Characters vs Compression)

Para la siguiente parte del estudio se tomará un set de 6 categorías y de ellas se extraerán sus P-Values y Effect Sizes donde la hipótesis nula es “Shrinko 8 no tiene efecto sobre el proyecto”. Separaremos los datos por recurso en los 6 sets, marcando un antes (Pre-minificación) y un después (Post-minificación más configuraciones).

En los resultados se encontró un P-value increíblemente pequeño, lo cual elimina la hipótesis nula, pero lo interesante son los valores del Effect-Size, corroborando que podemos descartar el análisis de tokens para los experimentos de la herramienta ya que son muy bajos, mientras que en los otros recursos presentan un módico valor, diciéndonos que el efecto de la herramienta en los datos es significante.

Minification

P-values

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	1.2524e-53	4.6366e-14	1.4302e-07	1.8337e-04	1.2937e-04	8.4823e-61
Characters	2.5513e-66	2.6976e-19	1.6163e-09	5.9605e-08	5.9605e-08	3.9935e-93
Compression	1.7350e-66	3.9285e-19	1.6107e-09	5.9605e-08	5.9605e-08	1.1968e-92

Effect Size

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	0.1115	0.0411	0.1220	0.0768	0.1232	0.0258
Characters	0.6352	0.5193	0.6585	0.5904	0.6976	0.4838
Compression	0.4552	0.3498	0.4648	0.4112	0.4880	0.3392

Focus Tokens

P-values

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	1.4466e-63	2.8874e-17	3.2785e-09	3.7940e-05	2.6300e-05	2.3185e-84
Characters	2.5503e-66	2.7022e-19	1.6206e-09	5.9605e-08	5.9605e-08	5.8139e-93
Compression	1.7364e-66	5.7530e-19	1.6147e-09	5.9605e-08	5.9605e-08	1.1913e-92

Effect Size

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	0.1590	0.0681	0.1712	0.1136	0.1920	0.0457
Characters	0.6332	0.5166	0.6564	0.5824	0.6960	0.4799
Compression	0.4524	0.3471	0.4598	0.4128	0.4848	0.3360

Focus Characters

P-values

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	1.2524e-53	4.6366e-14	1.4302e-07	1.8337e-04	1.2937e-04	8.4823e-61
Characters	2.5485e-66	2.6990e-19	1.6238e-09	5.9605e-08	5.9605e-08	2.7539e-93
Compression	2.5144e-66	5.7421e-19	1.6107e-09	5.9605e-08	5.9605e-08	1.1948e-92

Effect Size

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	0.1115	0.0411	0.1220	0.0768	0.1232	0.0258
Characters	0.6522	0.5331	0.6880	0.6032	0.7072	0.4998
Compression	0.4462	0.3431	0.4486	0.4000	0.4800	0.3333

Focus Compression

P-values

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	1.2524e-53	4.6366e-14	1.4302e-07	1.8337e-04	1.2937e-04	8.4823e-61
Characters	5.7831e-66	4.6736e-19	1.6175e-09	5.9605e-08	5.9605e-08	1.3225e-92
Compression	1.7333e-66	3.9234e-19	1.6091e-09	5.9605e-08	5.9605e-08	8.2442e-93

Effect Size

Resources	[Action-Adventure]	[Adventure-Visual]	[BeatEmUp]	[Gun-FPS]	[Hacks]	[Misc]
Tokens	0.1115	0.0411	0.1220	0.0768	0.1232	0.0258
Characters	0.6127	0.4987	0.6314	0.5632	0.6688	0.4573
Compression	0.4583	0.3514	0.4665	0.4176	0.4848	0.3420

Tabla 1: P-Values y Effect Size de las configuraciones.

Utilizando el dataset completo se trató de ver si se obtienen valores más regulares con los que se pueden sacar conclusiones. El P-value no es mostrado debido a que en todos los casos este dio 0, completamente negando la hipótesis nula. La compresión y los caracteres presentan un Effect-size que es similar en todas las configuraciones.

Effect Sizes, Dataset

Recursos	Minification	Focus Tokens	Focus Characters	Focus Compression
Tokens	0.0469	0.0740	0.0469	0.0469
Characters	0.5308	0.5281	0.5481	0.5054
Compression	0.3590	0.3557	0.3519	0.3615

Tabla 2: Effect Sizes del Dataset de proyectos.

Se puede concluir que la especialidad de la herramienta se centra en la reducción de caracteres de tal forma que está, siendo extremadamente agresiva, es capaz de reducir la compresión, siendo que estas dos van de la mano ya que la compresión depende de cómo son utilizados los caracteres dentro del. Se puede afirmar esta dependencia debido a que solo somos capaces de afectar la cantidad de tokens y caracteres durante la escritura del código de manera directa, en otras palabras, estos dos recursos son independientes, aunque relacionados, y debido a la falta de control que tenemos de la compresión, estos son dependientes de los caracteres.

Debido a estas razones, utilizaremos la herramienta Shrinko8 para la evaluación de código con respecto a su uso de caracteres y de compresión de ser necesario.

5.3.3 Minificación de Comentarios.

PICO-8, como dicho a lo largo de este texto, lo ideal es limitar los comentarios al mínimo, pero el programador tiene que elegir entre entendimiento del código o eficiencia en la utilización de recursos. Se quiere ver, dentro del dataset, varios datos estadísticos con respecto a los comentarios ya que son los más fáciles de eliminar sin tener que destruir la legibilidad del código.

Se minificará el dataset con focus en caracteres, y utilizando todas las configuraciones que nos permitan limitar la minificación a solo los comentarios. Después, tomaremos la cantidad de caracteres del dataset y la de la minificación sin comentarios para obtener la diferencia, la cual será la cantidad de caracteres que son comentarios por proyecto:

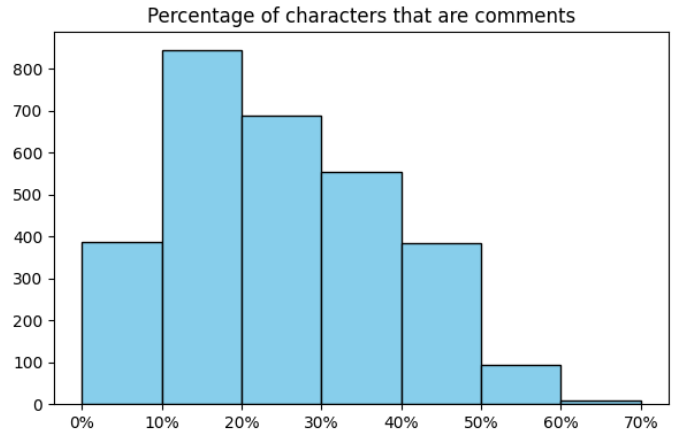

Fig 18: Porcentaje de Comentarios por Proyecto

De 2966 proyectos los caracteres que fueron comentarios nos dieron una media de 24.6%, una mediana de 23.0%, un máximo de 72.96% y un mínimo de 1.0%. 4 proyectos no fueron listados en el análisis debido a errores y resultados negativos o 0. Estos porcentajes están listados de acuerdo con el límite de caracteres, siendo 100% la máxima cantidad de caracteres posibles en un cartucho de PICO-8.

La media es de un total de 16,121 caracteres, la palabra promedio, asumiendo el lenguaje utilizado siendo el inglés, es de un total de 5.7, lo que nos da alrededor de 2,828 palabras. Varios proyectos utilizan los comentarios como método de documentación, aunque también existen proyectos han sido minificados ya sea por problemas de compresión u otras razones y, a lo menos en el dataset, carecen de comentarios. Una forma de verificar los intentos de minificación son la falta de saltos de línea y tabulaciones, siendo estos los más fáciles de modificar.

5.4 Conclusión del experimento

La utilización de la herramienta Shrinko8 mostro las relaciones entre recursos y sus capacidades reductivas, aunque en cuanto a los tokens, revisando los códigos modificados por la minificación, esta solo presenta el reemplazo de la función PRINT() por el atajo proporcionado por PICO-8. Aun así, su coeficiente relacional entre sí y los otros dos recursos es razonablemente alto.

En cierto sentido, cada utilización de token requiere de cierta cantidad de caracteres, como paréntesis. Aunque solo el primero cuenta como token, este tiene que estar cerrado para funcionar, por tanto, todo lo que se encuentra dentro de este se puede considerar como los caracteres necesarios del token, a lo menos 2 en cualquier caso. Viendo esto, existe una relación estrecha entre estos dos tal que por cada token hay a lo menos 1 carácter a menos que estos sean comentarios, nuevas líneas, tabulaciones o espacios.

6 Ejemplo de evaluación de código

Para finalizar este documento, se quiere aplicar lo visto con anterioridad a un proyecto desarrollado en PICO-8 tomando nota del estado del código, practicas presentadas, prácticas que se pueden implementar y la utilización de la herramienta Shrinko8 para alcanzar una óptima utilización de recursos y concluir si el proyecto presenta un código ideal para los propósitos por los que fue diseñado o la optimización de recursos da espacio para añadir funcionalidades que completen a la aplicación.

Cabe destacar que anteriormente existía el plan de realizar una métrica de evaluación de código la cual, desarrollando una herramienta automática para fines de recolección de información, se daría puntuación al proyecto y dar recomendaciones pertinentes, pero una puntuación tendría la dificultad de trabajar con las posibles inconsistencias presentadas en el código y que no sea considerada debido a su subjetividad; la idea principal de la memoria es traer prácticas de programación y optimización de PICO-8 a un ambiente científico y estudiar sus efectos en los proyectos, no estandarizar el código. Shrinko8 ya presenta capacidades de ‘Linting’, el chequeo automático del código fuente por errores programáticos y sintácticos [39], por lo cual esta utilidad queda cubierta por un programa ya existente. Es por estas razones que se decidió la evaluación de código de forma manual y que, idealmente, este fuese a ser utilizado como plantilla o guía para futuras revisiones.

Para el ejemplo utilizaremos un juego presentado actualmente en la página oficial de la plataforma llamado “The Carpathian” [40], el autor del cual no tiene muchos conocimientos de programación, citando tutoriales y mencionando que “La mayoría del código vino” de estos. Las razones de elegir este proyecto sobre otros son que el proyecto es mostrado en la página web principal de la plataforma, dando un valor más alto del proyecto como representante, además de que el proyecto es reciente, su última versión siendo de noviembre 2022, y que el código se encuentra a 71 tokens del límite; si se pueden optimizar una gran cantidad de tokens, entonces es posible modificar el proyecto ya sea implementando nuevas características o mejorando las existentes. Por último, el autor de este proyecto, como muchos otros dentro de la comunidad, se puede categorizar como un desarrollador aficionado, lo cual lo hace un excelente candidato como representante para el ejemplo de la evaluación.

6.1 Observaciones de la estructura del código

Algunas prácticas que se pueden considerar erróneas más recurrentes dentro del código son la utilización de palabras claves para nombrar variables de tablas y la utilización de condicionales en ciertas secciones de código de forma excesiva que podrían ser simplificadas a funciones utilizando tablas de valores llamadas por indexing. Un ejemplo seria la función Spawnen(); crea enemigos tomando coordenadas y el tipo de enemigo por id; lo erróneo está en la inicialización de variables de la tabla que pasara a ser el objeto Enemigo dentro de condicionales que utilizan el id del enemigo a crear para acceder a ella, en vez de crear tablas por antemano que contengan cada característica de los enemigos y luego llamar a esta tabla para la generación de nuevos enemigos. Ambas practicas resultan en la necesidad de reestructurar el código para su utilización optima.

6.2 Utilización de Tokens

Se presentará una tabla conteniendo las prácticas de tokens presentes y no presentes dentro del proyecto y las discutiremos.

Practica	En código	Notas
Default Arguments	×	No hay utilización visible de esta práctica dentro del código.
Asignación con comas	×	Varias asignaciones de variables en serie presentes en el código se beneficiarían de esta práctica.
Operaciones matemáticas	✓	Todos los valores están escritos y las operaciones matemáticas presentes son tales que son necesarias y no pueden ser representadas de otra forma.
Tablas de Lua	×	Uso excesivo de las tablas al momento de asignar valores de variables; No existe inicialización de tablas en el código como tal, gastando varios tokens.
Array Indexing	✓	Aunque el uso de asignación de valores de tablas es utilizado de forma errónea, debido a la cantidad de datos que tiene que manejar cada tabla, es ideal usar propiedades de tabla que arrays
Logical Short Circuiting	×	La utilización de operadores lógicos se limita a la utilización de igualdad (==). Aunque estos son mejorables.
Calling functions with strings or tables	×	Existen algunas funciones con un solo argumento, por tanto, su utilización es posible, aunque no critica en diferencia al resto de las practicas.
Parsing data from String literals	×	Varias partes del código se verían beneficiadas de esta práctica. En la sección anterior se habló de un ejemplo de mala práctica dentro del código y su solución, pero podríamos declarar esta tabla utilizando esta práctica en vez de la forma convencional al tener caracteres de sobra.

Tabla 3 Evaluación de Utilización de Practicas de Tokens

El total de tokens utilizados es del 99.1%; por lo visto en la tabla anterior, es posible la reducción del recurso de tal forma que se puedan crear nuevas funciones, características o sistemas dentro del proyecto.

6.3 Utilización de Caracteres y Compresión

Analizaremos varios aspectos del código con respecto a sus caracteres y su compresión. En este caso, también hablaremos de como Shrinko8 interactúa con el proyecto y si sus funciones son tales que optimizan el uso de recursos de este.

Utilizando 59.7% de los caracteres disponibles, el código, en este tema específico, muestra ciertos aspectos mejorables como la eliminación de partes de código dejadas en comentarios, tabulaciones y entrelineas. Por el lado positivo, los nombres de las variables son cortos y descriptivos, siendo la minoría los casos en los que no se cumplen estos criterios.

La compresión del programa es de 77%, lo cual, si fuésemos a realizar cambios en el programa y adiciones de características, como más enemigos, modos de juego o niveles, puede que se convierta en un problema al momento de querer exportar el programa a p8.png al llegar al tope del recurso. Utilizaremos Shrinko8 para encontrar si existe mejora de compresión al minificar de forma mínima (sin cambio de nombre de variables o eliminación de nuevas líneas) comparándola con una configuración que realiza las prácticas en la sección 4.2.1 en la parte de caracteres excepto la eliminación de nuevas líneas.

Recurso	N° de	% de
Tokens	8064	98.44%
Chars	31209	48%
Compressed	9353	60%

Tabla 4 Minificacion Mínima de The Carpathian

Recurso	N° de	% de
Tokens	8064	98.44%
Chars	25432	39%
Compressed	8525	55%

Tabla 5 Minificacion Practica de The Carpathian

Sin la necesidad de modificar el código de manera destructiva (Mínima, Tabla 4), el tamaño del código es capaz de ser comprimido utilizando un 17% menos del espacio máximo proporcionado, mientras que la minificación siguiendo las prácticas de caracteres (Practica, Tabla 5) lo disminuye un tanto más, pero en una relación menor entre pasos. Por otro lado, los caracteres bajan de forma constante lo cual puede significar una gran ayuda si es que llegásemos a este límite.

Como conclusión, si fuesen a faltar recursos de esta índole, lo más conveniente sería realizar una minificación de forma mínima y si al final del proyecto hay un margen menor en el cual la compresión necesaria es marginal, es posible utilizar la minificación utilizando prácticas.

6.4 _ENV en el proyecto

La utilización de _ENV puede optimizar los tokens de código de manera significativa, pero se debe estudiar la estructura del código para ver si es necesario su inclusión.

Varias partes del código utilizan una única tabla para el almacenamiento de variables en ambientes locales, como funciones, para la creación de objetos, por lo cual es posible utilizar _ENV para aligerar la utilización de tokens y de caracteres. Aun así, el código requiere de una reestructuración masiva debido a ciertos problemas como la utilización de palabras clave, como SPR, en funciones de las mismas tablas que queremos modificar; sin esto, es imposible utilizar _ENV.

6.5 Cartridge Chaining en el Proyecto

En los proyectos pueden desarrollar nuevas características utilizando Cartridge Chaining gracias a la capacidad incrementada del programa, pero también existe el caso que solo un cartucho cubra todas las necesidades presentes.

Si se fuese a aligerar la carga de recursos sería posible, entre otras cosas, la creación de nuevos enemigos o patrones de comportamiento, otros tipos de ataque que el jugador puede realizar, nuevos objetos y niveles. Aun así, en el caso de Cartridge Chaining, los usos que se podrían dar a los nuevos recursos serían expansiones al juego, generación de objetos y enemigos en lugares aleatorios y con propiedades únicas, pasándolos a la memoria o un cartucho capaz de realizar modificaciones a los niveles existentes; un creador de niveles por así decirlo.

6.6 Conclusión de la evaluación

El código no se encuentra optimizado en muchas áreas y es propenso a la realización de malas prácticas que, además de utilizar una gran cantidad de recursos, nos obligan a realizar cambios significativos a la estructura del programa con el objetivo de disminuir la utilización de recursos. Después de haber realizado esto, sin embargo, es posible la creación de nuevas características del programa utilizando los nuevos recursos disponibles y así mejorar las aptitudes del proyecto.

7 Conclusión y discusión

Este documento presenta varias técnicas y recordatorios de programación para cualquier usuario que los requiera en este ámbito de desarrollo. Es importante tener en cuenta la utilización de estas es dependiente de la estructura de código de cada proyecto y que no siempre serán utilizadas con la misma eficacia. Aun así, teniéndolas en cuenta al momento de programar significara una experiencia más homogénea en cuanto a la facilidad de realizar optimizaciones y el tener un sentido del peso de los recursos en las funciones a realizar, acostumbrándose a los límites de la plataforma.

Amenazas a la validez: Cambios a la sintaxis de PICO-8 que alteren algunas de las practicas presentadas o a la definición de recursos, aunque estos no ocurren con una alta frecuencia como se puede apreciar en el registro de versiones [22]. Algunas versiones pueden tener bugs en cuanto a la sintaxis, por lo que existe la posibilidad de que nuevas prácticas desarrolladas o incluso las escritas en este documento puedan ser “arregladas” en versiones siguientes, lo cual podría requerir reescritura del documento.

En cuanto al dataset, este es tal que la calidad de código en cada proyecto es altamente variable y puede no ser representativa del desarrollador promedio de la plataforma, además de la falta de proyectos que utilizan Cartridge Chaining, pero esto es debido a las limitaciones de la plataforma impuestas por los creadores. Aun así, estos proyectos son representativos de la comunidad y este es el único trabajo de esta índole sobre PICO8, entonces, por el momento, el trabajo mas significativo hasta la fecha sobre la plataforma.

Trabajo futuro: El trabajo realizado se puede considerar de investigación más que de práctica, por tanto, si es posible, la ideación de prácticas originales capaces de optimizar la utilización de recursos dentro de la plataforma sería el propósito final de esta memoria de título, además de un entendimiento profundo de la plataforma y sus diferentes casos de uso.

Actualmente, los desarrolladores de PICO-8 han creado un ‘Computador de Fantasía’ llamado Picotron [41]. Más cercano a un sistema operativo, tiene por objetivo la creación multimedia en un ambiente minimalista. Los programas tienen mayor capacidad, el ambiente es completamente personalizable, más colores, opciones gráficas, entre otros. Lo más importante en este caso es la compatibilidad de proyectos y syntax de PICO-8, haciendo la transición a la nueva plataforma bastante fácil. Esto puede incluir a las practicas mencionadas en el documento, pero esto requiere de más pruebas. También existe el caso en el que se puedan crear más practicas gracias a los nuevos sistemas presentes, del cual aún no existe

documentación como tal. Al querer desarrollar en la plataforma, realizar este mismo tipo de documentación para el nuevo sistema seria también un objetivo a futuro. Como ahora hablamos de un sistema operativo, Stress Testing también sería ideal; los límites de la plataforma, cuantos programas puede correr, de qué tamaño son y su optimización como plataforma de trabajo y sistema. No solo nos estamos preocupando de técnicas de programación, lo interesante también seria ver las utilidades que puede tener esta nueva plataforma a lo largo del tiempo, y si es capaz de reemplazar PICO-8 o a lo menos asimilar está dentro de su ambiente.

8 Bibliography

[1]J. "Zep White, “PICO-8 Fantasy Console,” *www.lexaloffle.com*. https://www.lexaloffle.com/pico-8.php?page=faq (accessed Oct. 25, 2023).
[2]Ericb", “Into Ruins,” *www.lexaloffle.com*, Oct. 27, 2022. https://www.lexaloffle.com/bbs/?pid=119614 (accessed May 10, 2024).
[3]J. "zep White, “PICO-8 Manual,” *www.lexaloffle.com*. https://www.lexaloffle.com/dl/docs/pico-8_manual.html
[4]I. Lyman, “Is software getting worse?,” *stackoverflow.blog*, Dec. 25, 2023. https://stackoverflow.blog/2023/12/25/is-software-getting-worse/ (accessed Jun. 05, 2024).
[5]S. S. LeBlanc, “PICO-8-Token-Optimizations,” *GitHub*, Jul. 03, 2024. https://github.com/seleb/PICO-8-Token-Optimizations (accessed Mar. 14, 2024).
[6]JScott, “Writing Good Code,” *www.lexaloffle.com*, Sep. 22, 2015. https://www.lexaloffle.com/bbs/?tid=2508 (accessed May 10, 2024).
[7]J. Scott, “Nightly Build,” *blog.jvscott.net*, 2018. https://web.archive.org/web/20180325175211/http:/blog.jvscott.net/tagged/best-practices (accessed Oct. 21, 2023).
[8]“Lua: About,” *Lua.org*, 2011. https://www.lua.org/about.html (accessed Jun. 2024).
[9]R. Ierusalimschy, L. H. de Figueiredo, and W. Celes, “The Evolution of Lua,” PUC-Rio, Rio de Janeiro, Brazi. Accessed: Apr. 03, 2024. [Online]. Available: https://www.lua.org/doc/hopl.pdf
[10]J. "Zep White, “Voxatron Development Diary 3,” *Lexaloffle.com*, Sep. 14, 2012. https://www.lexaloffle.com/bbs/?tid=1248 (accessed Jul. 06, 2024).
[11]J. Peitz, “PicoCAD,” *itch.io*, Feb. 25, 2021. https://johanpeitz.itch.io/picocad (accessed Jan. 05, 2024).
[12]C. Drum, “PicoCalc by Christopher Drum,” *itch.io*, 2021. https://christopherdrum.itch.io/picocalc (accessed Jan. 05, 2023).
[13]Luchak, “RP-8,” *itch.io*, 2023. https://luchak.itch.io/rp8 (accessed Jan. 05, 2024).
[14]electricgryphon", “Gryphon 3D Engine Library,” *www.lexaloffle.com*, Nov. 17, 2016. https://www.lexaloffle.com/bbs/?tid=28077 (accessed Jan. 05, 2024).
[15]E. G. Cota, “kikito/bump.lua,” *GitHub*, 2012. https://github.com/kikito/bump.lua (accessed Jan. 05, 2023).
[16]purple-pixels, “Easy Google Analytics event tracking for all your PICO-8 games.,” *GitHub*, Jan. 05, 2024. https://github.com/purple-pixels/pico8-google-analytics (accessed Sep. 06, 2024).
[17]S. Hocevar, “z8lua,” *GitHub*, 2017. https://github.com/samhocevar/z8lua (accessed Jan. 05, 2024).
[18]“Tokens,” *PICO-8 Wiki*. https://pico-8.fandom.com/wiki/Tokens (accessed Nov. 23, 2023).

[19]“P8PNGFileFormat,” *PICO-8 Wiki*. https://pico-8.fandom.com/wiki/P8PNGFileFormat (accessed Nov. 23, 2023).
[20]“PICO-8 Wiki,” *pico-8.fandom.com*. https://pico-8.fandom.com/wiki/Pico-8_Wikia (accessed Nov. 23, 2023).
[21]J. "Zep White, “PICO-8 Fantasy Console,” *www.lexaloffle.com*. https://www.lexaloffle.com/pico-8.php (accessed 2024).
[22]“PICO-8 Changelog,” *Lexaloffle.com*, 2024. https://www.lexaloffle.com/dl/docs/pico-8_changelog.txt (accessed Apr. 07, 2024).
[23]Jeblad, “Lua Best Practice,” *MediaWiki*, Dec. 20, 2017. https://www.mediawiki.org/wiki/Help:Lua/Lua_best_practice (accessed Oct. 13, 2023).
[24]R. Ierusalimschy, “Programming in Lua: Chapter 13,” *www.lua.org*, 2003. https://www.lua.org/pil/13.html (accessed Feb. 14, 2024).
[25]“Lua 5.4 Reference Manual; 2.4 Metatables and Metamethods,” *Lua.org*, 2020. https://www.lua.org/manual/5.4/manual.html#2.4 (accessed May 2024).
[26]R. Ierusalimschy, “Programming in Lua: Chapter 14, The Enviroment,” *www.lua.org*, 2003. https://www.lua.org/pil/14.html
[27]2DArray", “Optimizing Character-Count for Tweetcarts,” *demoman.net*. https://demoman.net/?a=optimizing-for-tweetcarts (accessed Dec. 11, 2023).
[28]Viggles", “A Compressed Size Limit Rant,” *www.lexaloffle.com*, Apr. 11, 2016. https://www.lexaloffle.com/bbs/?tid=3205 (accessed Apr. 13, 2024).
[29]Wikipedia Contributors, “Shannon’s Source Coding Theorem,” *Wikipedia*, May 22, 2022. https://en.wikipedia.org/wiki/Shannon%27s_source_coding_theorem (accessed Jun. 29, 2024).
[30]Wikipedia Contributors, “Feature Creep,” *Wikipedia*, Sep. 07, 2021. https://en.wikipedia.org/wiki/Feature_creep (accessed Jun. 29, 2024).
[31]kleril, “Cartridge Chaining and File I/O Example,” *www.lexaloffle.com*, Aug. 06, 2015. https://www.lexaloffle.com/bbs/?tid=2238
[32]“POOM by freds72, Paranoid Cactus,” *itch.io*. https://freds72.itch.io/poom (accessed Mar. 13, 2024).
[33]“Journey to POOM - Devlog,” *itch.io*. https://freds72.itch.io/poom/devlog/241700/journey-to-poom (accessed Mar. 13, 2024).
[34]R. Petrov, “Making of PICOCHAK,” *Medium*, May 04, 2020. https://medium.com/@megus/making-of-picochak-be39d383d87f (accessed Jun. 02, 2024).
[35]Gio, “Jinja2 as a Pico-8 Preprocessor,” *Giovanh.com*, Dec. 11, 2022. https://blog.giovanh.com/blog/2022/12/11/jinja2-as-a-pico-8-preprocessor/ (accessed May 21, 2024).
[36]F. Bueno, “Awesome-PICO-8,” *GitHub*, 2015. https://github.com/pico-8/awesome-PICO-8 (accessed Dec. 28, 2023).
[37]thisismypassport, “Shrinko8,” *GitHub*, 2023. https://github.com/thisismypassport/shrinko8 (accessed Jun. 08, 2024).
[38]C. Andrade, “The P Value and Statistical significance: Misunderstandings, explanations, challenges, and Alternatives,” *Indian Journal of Psychological Medicine*, vol. 41, no. 3, p. 210, 2019, doi: https://doi.org/10.4103/ijpsym.ijpsym_193_19.
[39]G. Saladino, “What Is Linting + When to Use Lint Tools,” *Perforce Software*, 2019. https://www.perforce.com/blog/qac/what-is-linting (accessed Apr. 29, 2024).
[40]Trog, “The Carpathian,” *Lexaloffle.com*, Aug. 17, 2022. https://www.lexaloffle.com/bbs/?pid=the_carpathian#p (accessed Jul. 09, 2024).
[41]J. "Zep White, “Picotron by Lexaloffle,” *Lexaloffle.com*, 2024. https://www.lexaloffle.com/picotron.php (accessed Jul. 14, 2024).

[42]”slainte”, “Using _*env* in PICO-8”, *www.lexaloffle.com,* Aug. 25, 2022. https://www.lexaloffle.com/bbs/?tid=49047.

Anexo A: Resumen de practicas

Practice and Main Sources	Summary	Advantages	Constraints
_ENV [26] [42]	Utilizing the Lua article, you can change the current enviroment and make a table the new "global", so every new variable actually belongs to the table.	Reduction of tokens and characters by erasing the need of casting the table whenever having to call values	There exist several setups for the utilization of _ENV, and the programmer needs to decide on which to use depending on their necessities and if they really reduce more tokens than adding them.
Default Arguments [5]	Utilize the function available with the least amount of functions or create one such as it's most repeated argument along the code is within the function (Ex. Foo(5) is 6 times along the code, we can make it so Foo() is the same as Foo(5))	Token Reduction to the most used value of the argument of a function.	Only works if the function has a single argument, due to syntaxis issues when there's two or more.
Comma Asignation [5]	Multiple declarations of variables in the same line of code utilizing commas to separate each variable and then each value of said variables	Token Reduction by each '=' symbol, no line break and less bloated code	None percievable
Math operations [5]	All static values should be used instead of letting the code do the work. In other words, when doing mathematical operations between two static elements (Ex. 2x3), it is better to just write the result in the code (Ex. 6).	Token reduction and legibility by reducing the number of elements per operation.	None percievable
Table initialization [5]	Use tables when necessary and initialize their properties instead of declaring them as the table needs it. Otherwise, create local variables instead.	Reduce tokens when declaring table properties and a better arrangement of the written code by having all properties in a single block of code.	None percievable

Array Indexing [5]	Utilize the methods of array indexing or call to property depending on the amount of properties and calls that are necessary. In the first case, creating requires 1 token per property and calls need 2 at minimum, while on the latter it requires 3 per property and 1 per access.	Token reduction when choosing the correct option for the case scenario.	Requires a little bit of math to come out with which would be the best option to use.
Logical Short Circuiting [5]	Utilizing logical operators, value assignation utilizing functions can be heavily shortened instead of using conditional statements.	Token reduction but also character reduction as a side effect due to how expensive conditional statements are character wise.	Legibility for coders who are not knowledgeable of Lua's logical operators.
Calling Functions with strings or tables [5]	You can call a function without () if they use a string or a table.	Single token reduction	This only works when the function receives a single argument.
Parsing data from String literals [5]	Instead of having static values on a table, use a string literal and then parse it into an array or such.	Multiple token reduction per static value	The string literal must be sufficiently big so that the parsing doesn't take more tokens than the table equivalent.
Line Breaks [27]	Each of these reduces the empty space according to their name; by reducing new lines according to certain rules, renaming functions and variables to shorten them as much as possible or reordering if-statements respectively.	Massive Character reduction	These practices are the most destructive for the developer's code, making it highly illegible.
Function renaming [27]			
If-statements [27]			
Cartridge Chaining [31]	Utilize multiple cartridges (aka. PICO-8 files) to create a bigger project by adding resources and separating functions, elements and data by cartridge as needed.	Basically illimited resources for any kind of project.	Adding much more complexity to the project itself.

Anexo B: Metatablas y Metametodos de Lua

Una metatabla es una tabla que es utilizada en la función ‘setmetatable(a,mt)’ donde ‘a’ es una tabla y ‘mt’ la metatabla utilizada como base para otorgarle características a la tabla ‘a’. Los metametodos son artículos o eventos que se pueden modificar por metatabla, los cuales definen el comportamiento de operaciones aritméticas y relacionales, de acceso de tablas y propias de ciertas librerías [25]. En el siguiente script de código, se observa la declaración de Set y sus funciones utilizando una tabla, la declaración de una tabla dentro de Set que será la metatabla usada por estas funciones al momento de crear sets y así otorgar características como es el caso del ejemplo la utilización del operador de adición para realizar la función de unión.

```
Set = {}
Set.mt = {}

function Set.new (t)
  local set = {}
  setmetatable(set, Set.mt)
  for _, l in ipairs(t) do set[l] = true end
  return set
end

function Set.union (a,b)
  local res = Set.new{}
  for k in pairs(a) do res[k] = true end
  for k in pairs(b) do res[k] = true end
  return res
end
```

```
Set.mt.__add = Set.union

s1 = Set.new{10, 20, 30, 50}
s2 = Set.new{30, 1}
s3 = s1 + s2
Set.print(s3)  --> {1, 10, 20, 30, 50}
```

Fig Anexo B: Ejemplo de Uso de Metatablas y Metametodos

De esta forma es posible otorgarles propiedades a las tablas y darles un comportamiento similar a objetos en otros lenguajes, con lo cual es posible mejorar la estructura del código y otorgar mayores posibilidades de personalización a su escritura.

Anexo C: Representación matemática de prácticas de la sección 4.2

1.- Default Arguments;

$$\exists\, x\,, \forall foo^{n}(x), t_1 : 5 \;<\; n \;\wedge\; x \;=\; [x_1, x_2, \ldots, x_m] \;\wedge\; n, m, t_1 \in N$$
$$\rightarrow\; \exists\, foo^{n}(),\; t_2 :\; foo() \;=\; foo(x) \;\wedge\; t_1 \;>\; t_2 \;\wedge\; t_2 \in N$$

Traduccion: Si existe un x para toda funcion $foo(x)$ que se repita n veces, con un elemento t_1 que representa los tokens del codigo, tal que, n sea mayor a 5, x un conjunto de m elementos donde n,m y t1 pertenecen a los numeros naturales, entonces existe una funcion foo() y un elemento t2 tal que foo() es equivalente a foo(x) y t1 es menor a t2, siendo que t2 pertenece a los naturales.

Una forma mas facil de verlo es con la ecuacion:

$$3n + 4 > 2n + 9 \;/\; -2n, -4$$
$$n > 5$$

La inegualdad representa el numero de tokens de los ejemplos 1 y 2 del informe, n siendo el numero de veces que la funcion se repite con el mismo argumento. En el primer ejemplo se tienen 3 tokens por llamada a la funcion mas los 4 tokens para la declaracion de la funcion, y en el segundo son 2 tokens por llamada, pero la declaracion para acomodar el argumento default requiere 9.

5.- Array Indexing;

Teniendo un elemento t que representa el numero de veces que se haya llamado al elemento de un array, podemos representar la asignacion de un array de dos formas:

$$Array \;=\; \{x_1, x_2, \ldots, x_{n-1}, x_n\} \;:\; n \in N \;\rightarrow Array[n-1] \;=\; x_n \quad (1$$
$$Array \;=\; \{x_1 = x_1, x_2 = x_2, \ldots, x_{n-1} = x_{n-1}, x_n = x_n\} \;:\; n \in N \;\rightarrow Array.x_n = x_n \quad (2$$

En la primera ecuacion tenemos que la declaracion toma $3 + n$ tokens y la llamada toma 3 tokens, entonces si se llama t veces tenemos que el total de tokens en (1 es de $3 + n + 3t$. En la segunda ecuacion tenemos que la declaracion toma $3 + 3n$ tokens y la llamada toma 2 tokens, entonces si se llama t veces tenemos que el total de tokens en (2 es de $3 + 3n + 2t$.

Entonces tenemos que para decidir que formato utilizar, tenemos la desigualdad:

$$3 + n + 3t < 3 + 3n + 2t$$

Tal que si se cumple, utilizaremos (1, caso contrario utilizamos (2, pero podemos resolver esta desigualdad:

$$3 + n + 3t < 3 + 3n + 2t \quad /- n, -3, -2t$$
$$0 + 0 + t < 0 + 2n + 0$$
$$t < 2n \quad / \cdot 1/2$$
$$t/2 < n$$

Entonces como conclucion tenemos:

$$\text{Array} = \begin{cases} (1 & \text{if } t/2 < n \\ (2 & \text{if } t/2 \geqslant n \end{cases}$$

6.- Logical Short Circuiting; En lua se tiene que:

$$\exists\, x, \text{foo}(), \text{bar}() : x = \begin{cases} \text{foo}() & \text{if foo}() \notin \emptyset \\ \text{bar}() & \text{if foo}() \in \emptyset \end{cases} \Leftrightarrow x = \text{foo}() \vee \text{bar}()$$

Traduccion: existe un x y funciones foo() bar() tales que x es igual a foo() si este no pertenece al conjunto vacio, o x es igual a bar() si foo() pertenece al conjunto vacio, y esto es equivalente (en el ambiente de Lua) a que x sea igual a foo() o bar().