

Documentación casino

Proyecto Entornos de Desarrollo

Introducción	2
Metodología	2
Desarrollo del proyecto	2
Organización y gestión del proyecto	2
Menú principal	3
Tragaperras	4
Primera versión de la máquina tragaperras	4
Segunda versión de la máquina tragaperras	5
Tercera versión de la máquina tragaperras	6
Carrera de caballos	7
Fase 1	7
Fase 2	8
Fase 3	8
Ruleta	11
Fase 1 - Idea y variables	11
Fase 2 - Funciones y ejecución principal	12
Fase 4 - PrettyTable	14
Bingo	15
BlackJack	18
Rama cambios	21
Gestión de incidencias	21
Conclusiones	21
Bibliografía	22
Anexo	22

Introducción

Este documento resume el desarrollo del proyecto Casino. Casino es un programa desarrollado en Python, con el uso de diferentes módulos adicionales, que reúne una serie de juegos los cuales son accedidos por la terminal del Visual Studio Code. Además, usaremos Git para almacenar las diferentes versiones con las que estamos trabajando.

El objetivo del proyecto es crear un casino digital con una serie de juegos, donde se puede escoger una apuesta y ganar o perder dinero, según la suerte del jugador.

Metodología

La metodología que usamos fue plantear en un principio una serie de juegos que podrían formar parte de un casino digital. Tras esto cada programador elige uno y se ocupa de desarrollar ese mismo, intentando que dos programadores no tengan el mismo proyecto. Esto después se pone en común.

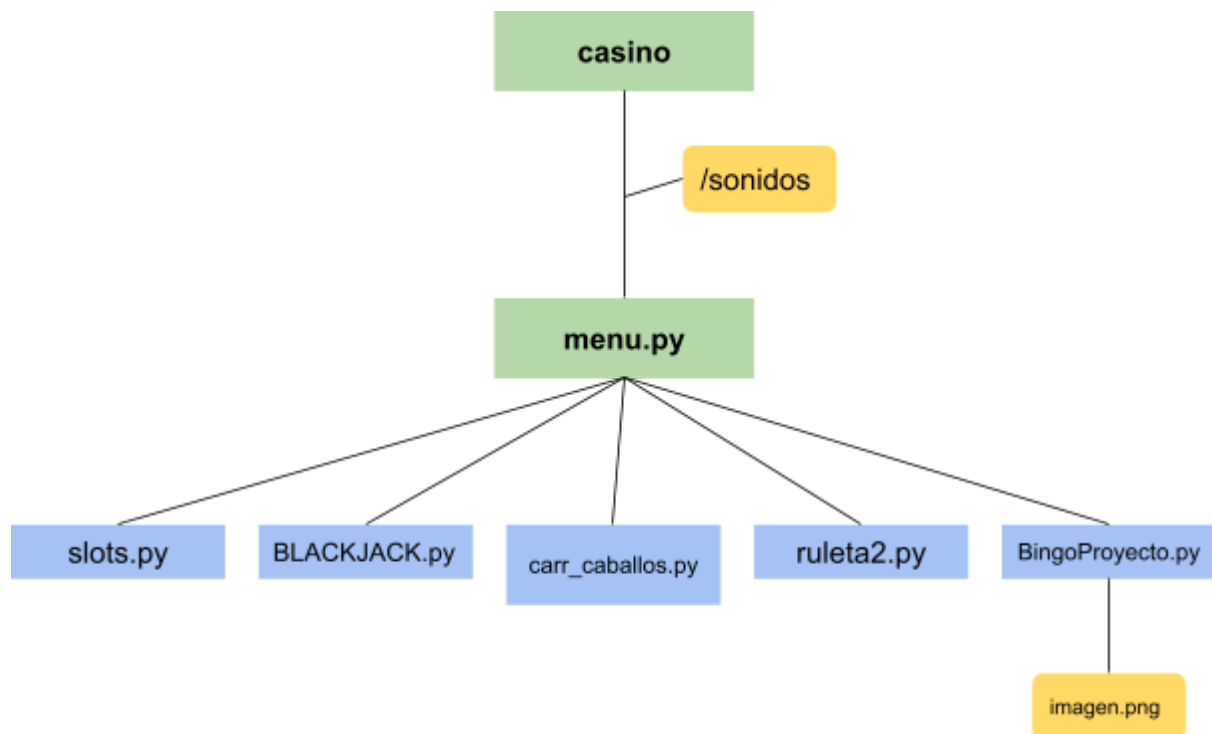
Por lo tanto, cada uno es responsable de que su juego funcione correctamente, esto se podrá después en común, creando un grupo de programas que serán accedidos por un programa principal de menú.

Desarrollo del proyecto

Organización y gestión del proyecto

El desarrollo del proyecto se ha realizado mayormente de forma individual, siendo cada programador el único responsable de su juego y parte del programa. Por lo tanto, el desarrollo se divide en el desarrollo individual de cada juego, mostrados a continuación.

Esquema de funcionamiento:



Menú principal

El programa principal importa sólo dos módulos; *playsound* para los efectos de sonido y *os* para limpiar el terminal cada vez que se cambia de escena. El funcionamiento del menú es bastante sencillo.

Este menú integra opciones para todos los juegos disponibles en el casino. Se accede a los juegos a través de un input que funciona tanto con el nombre del juego como con su número asignado. Para evitar los errores no hay un *try except* pero la función `opciones()` que recoge el input del usuario devuelve Falso si la opción escogida no se contempla en las opciones disponibles, si es verdadera lo que hace es importar el juego que está almacenado en la misma carpeta.

Las otras funciones, `visual_casino()`, `visual_juegos()` y `start()` simplemente sirven para ser llamadas y que muestren en pantalla el código ASCII, por la función `menu()`. La función `menu()`, llama primero a `visual_casino()`, después a `visual_juegos()`. Aquí hay un bucle que mientras `opciones()` devuelva un valor False vuelve a ser llamado hasta que la opción que se introduzca sea correcta.

En la corrección final de bugs e implementación de todos los juegos, el blackjack y el bingo no se ejecutaban correctamente con la importación, así que tuve que usar `sys.executable` para que corriesen.

Tragaperras

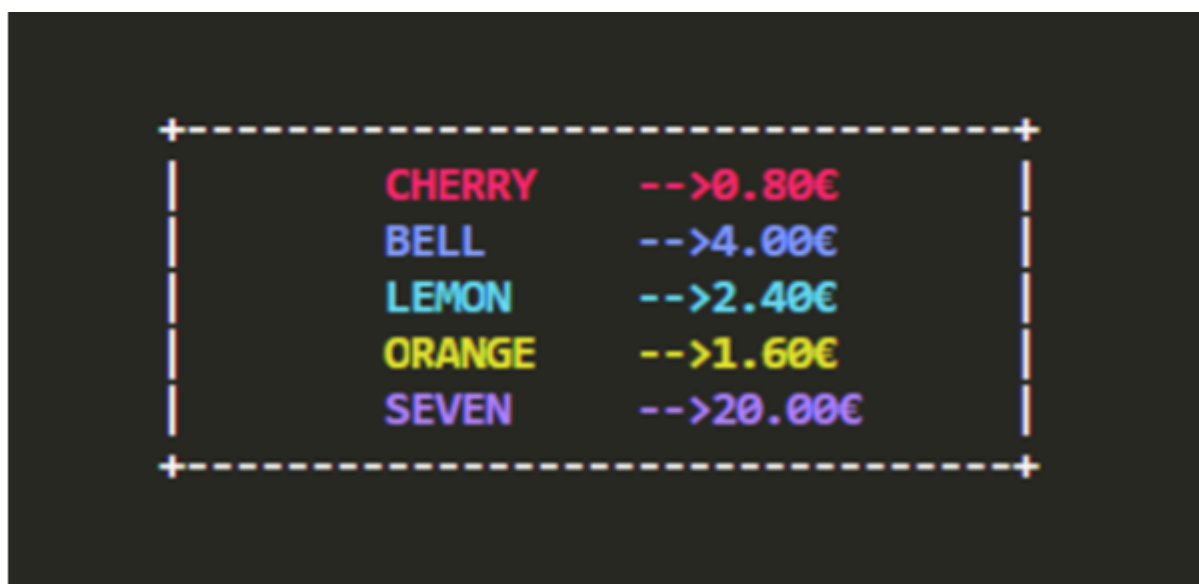
Como con el menú importa varios módulos; *os*, *playsound*, *time* y *random*.

Hay varias funciones también que simplemente muestran código ASCII en la terminal cuando son llamadas. Por una parte están las constantes, como el `dinero_inicial` y los ítems. Los ítems que aparecen en los slots de la máquina funcionan con un diccionario que el nombre llama al emoji que le corresponde. Después están las variables, el dinero y los diferentes slots, que se modifican con cada tirada. La función `jugar()` muestra la máquina tragaperras en pantalla y hace las tiradas. En `preguntarJugador()` funciona con un input, cuando el jugador introduce "s" continúa el juego, con "n" se para. Con `tirada()` simplemente se devuelve un valor aleatorio, mientras que en `puntuación()` dependiendo de el valor que salga en los slots se gana una cantidad diferente de dinero y se muestra por pantalla.

Las funciones se mantienen a lo largo de las diferentes versiones en el desarrollo de esta tragaperras, que he dividido el desarrollo en tres fases principales:

Primera versión de la máquina tragaperras

La primera versión de `slots.py` funciona como acabo de describir anteriormente. Cada vez que el jugador hace una tirada gana dinero o pierde según los resultados de cada slot, es la versión más sencilla y primitiva. En la primera versión estos eran los valores de cada resultado (Figura 1):



CHERRY	-->0.80€
BELL	-->4.00€
LEMON	-->2.40€
ORANGE	-->1.60€
SEVEN	-->20.00€

Figura 1.

Por lo tanto, si el resultado de la tragaperras era una línea de emojis de limones se sumaba 2.40€ al dinero que tenías en ese momento, y se mostraba en pantalla de la siguiente manera (Figura 2):

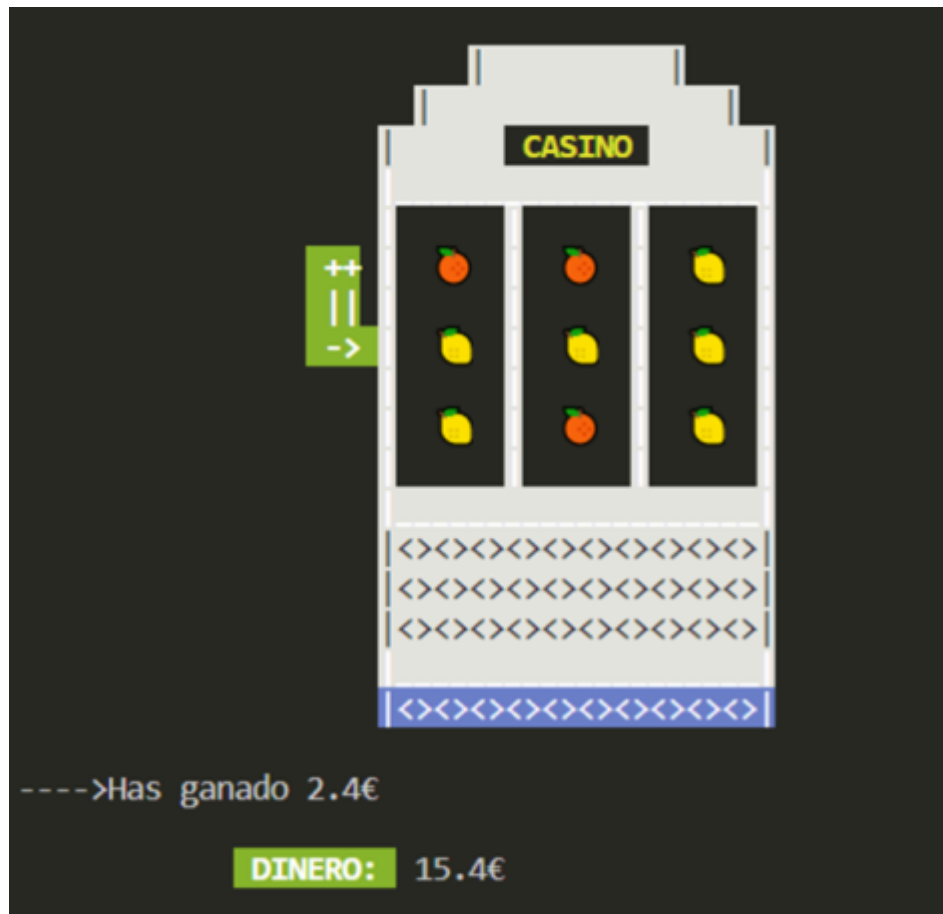


Figura 2.

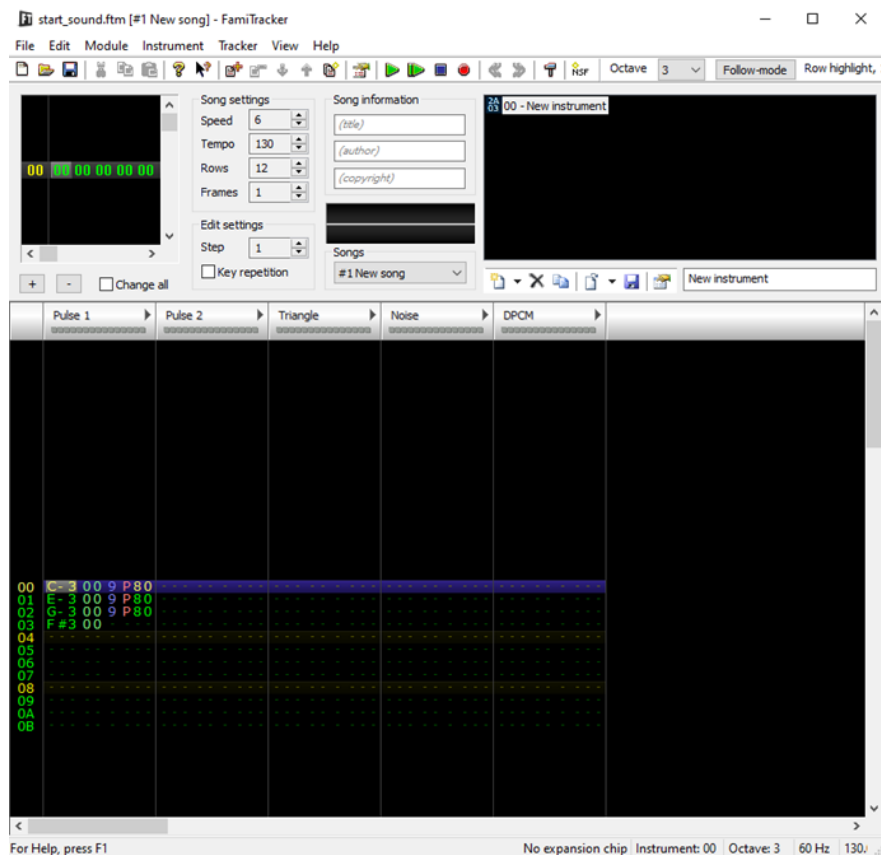
De nuevo, en cada tirada se limpia la terminal y se vuelve a imprimir de nuevo. En caso de terminar sin dinero la partida termina, y si no se quiere seguir tirando se muestra un mensaje de EXIT. El juego en este punto estaba completamente operable, pero las probabilidades de sacar una línea eran tan pocas que ganar era casi imposible, así que hice que las filas de encima y debajo contasen también y se almacenasen en variables (antes funcionaban simplemente con un `items_emoji[items[random.randrange(0,5)]]`).

Segunda versión de la máquina tragaperras

Esta versión cambia el funcionamiento de la máquina tragaperras, ahora se gana dinero también con las otras dos filas. Funciona tanto en vertical como en horizontal, además de las dos diagonales. No hay mayores cambios en el funcionamiento, las

ganancias de cada *item* siguen siendo las mismas. Si hay modificaciones en el código, que al existir más opciones es más compleja la función de puntuacion(), por lo que hay cambios en la forma en la que se recoge el resultado de esta en jugar(). Ahora si no ganas nada en la función puntuacion() el mensaje de 'Has perdido' es mostrado por jugar, ya que desde puntuación() daba error al existir tantas opciones. El código en este punto puede ser mejorado, ya que tiene estructuras que pueden reducirse.

Sin tener relación con la programación del juego, en este punto introduzco el módulo *playsound* que uso para hacer distintos sonidos: un sonido de inicio para la pantalla principal del juego, *start_sound.wav*; un sonido para cuando se gana una tirada, *bank.wav* y un sonido para cuando se pierde, *bonk.wav*. Estos sonidos se hacen con un programa externo, FamiTracker. FamiTracker es un programa que ya conocía de antes, se usa principalmente para producir música para la NES/Famicom, básicamente es un programa para hacer *chiptune*, como tantos otros que existen.



En este momento, a diferencia de anteriormente, perder es casi imposible, porque hay demasiadas posibilidades que siempre acaba ganando el dinero (comprobado tras múltiples intentos). Por lo tanto, en la versión última se añade la característica de las apuestas a la máquina tragaperras, que modifica el modo de juego.

Tercera versión de la máquina tragaperras

En esta versión además de importar un nuevo módulo, *time*, para que los cambios entre pantallas no sean tan bruscos, se modifica el modo de juego añadiendo apuestas a la

máquina tragaperras. Visualmente se añaden unas instrucciones en la pantalla de inicio y ahora los valores están disponibles siempre al jugar.

En esta versión es donde se trabaja más el código, se añaden diferentes *try except* para intentar pillar todos los errores, para esto se va probando todas las opciones posibles encontrando diferentes errores que daría el código al intentar operarlo de una determinada manera. Estos errores se buscan principalmente en la función de apuestas() que en las primeras versiones te dejaba apostar más dinero que el que se tenía o apostar cero euros. En jugar() todo continúa igual, sólo se añade un sleep(2), para que la pantalla de EXIT se pudiese leer ya que sino saltaba rápido a la siguiente pantalla. En puntuacion() cambia la manera en la que suma el dinero, ahora se tiene en cuenta la apuesta y se multiplica por el valor que le corresponde a cada ítem, después ese valor se suma al dinero, en caso de perder se pierde el dinero que se ha apostado.

```
#slot 3
if ((slotTresA=='CHERRY') and (slotTres=='CHERRY') and (slotTresB=='CHERRY')):
    win8=apuesta*0.80
elif ((slotTresA=='BELL') and (slotTres=='BELL') and (slotTresB=='BELL')):
    win8=apuesta*4.00
elif ((slotTresA=='LEMON') and (slotTres=='LEMON') and (slotTresB=='LEMON')):
    win8=apuesta*2.40
elif ((slotTresA=='ORANGE') and (slotTres=='ORANGE') and (slotTresB=='ORANGE')):
    win8=apuesta*1.60
elif ((slotTresA=='SEVEN') and (slotTres=='SEVEN') and (slotTresB=='SEVEN')):
    win8=apuesta*5.00
else:
    win8=0
dinero+=win8
if (win1==0 and win2==0 and win3==0 and win4==0 and win5==0 and win6==0 and win7==0 and win8==0):
    dinero-=apuesta
```

Lo más complicado de esta versión fue trabajar con la variable de apuesta y su función apuestas(), además de la suma y resta en esta parte. Aunque el código todavía puede ser optimizado, considero que es la versión final.

Carrera de caballos

La creación de este programa se puede dividir en 3 fases.

Fase 1

Para que el programa sea exactamente como lo veía en mi mente, lo primero era planificar cómo hacerlo. Pensé que podría hacer el programa en dos partes: La primera es todo el código, donde ocurren los cálculos y luego la animación. Consideraba que si primero hacía todo sobre los cálculos, luego me podría centrar en sólo hacer la interfaz gráfica, sin preocuparme de nada más,

Fase 2

El desarrollo del código del cálculo de las apuestas, pagos y probabilidades resultó muy sencillo. Decidí crear una clase llamada *carrera_caballos()*. Es una función en la que no se pasa ningún parámetro y devuelve el valor por el que se multiplica la apuesta. En caso de perder, devolvería 0. También puse que todos los caballos tuvieran las mismas probabilidades de ganar, por lo que simplemente puse que se eligiese como ganador uno de los 5 caballos.

```
def carrera_caballos():
    caballos = ['Juan', 'Speedy', 'Myke Tyson', 'Estorbo', 'Lentín']
    caballo_apostado = input('A que caballo deseas apostar? ')
    ganador = caballos[random.randint(0, 4)]
    print(ganador)
    if caballo_apostado == ganador:
        print('ganas')
        return 4.5
    else:
        print('pierdes')
        return 0

saldo = 2000
while True:
    apuesta = int(input('Indica tu apuesta'))
    saldo -= apuesta
    saldo += apuesta*carrera_caballos()
    print(saldo)
```

Fase 3

Ahora tocaba hacer la animación para que todo quedase más atractivo para el usuario. Mi idea era representar visualmente cómo los caballos se movían hasta el final de la pista. Comencé por buscar un banner para que aparezca en el terminal a qué juego se está jugando. Esta página es perfecta para lo que buscaba: <https://manytools.org/hacker-tools/ascii-banner/>

Cuando finalmente me puse a idear cómo hacer que los caballos se moviesen, me di cuenta que tal y como tenía hecho el código, resultaría muy lento hacerlo, porque tendría que hacer una animación distinta para cada ganador y además no quedaría tan interesante. Así que muy a mi pesar, tuve que rehacer casi todo el código.

Comenzé por crear una clase *Caballo()* que recibía el nombre del caballo y ponía su posición a 0. Con este nuevo sistema de posición, mi idea era cambiar todo el sistema que decidía el caballo ganador. Para esto primero se importa el módulo *os* para poder ir actualizando la pantalla. Todo este proceso se encuentra dentro de un bucle. En él aleatoriamente con cada ciclo se le suma 1 o 0 a la posición del caballo, por lo que en cada iteración del bucle cada caballo se mueve independientemente.

```
class caballo:
    def __init__(self, nombre):
        self.nombre = nombre
        self.posicion = 0

    def mover(self):
        self.posicion += random.randint(0, 1)
```

Luego con un simple bucle que escribe por consola un espacio por cada posición que tiene el caballo guardado en su clase. Por último un *time.sleep()* de 0.2 segundos para que la animación se vea a una velocidad buena y listo. El programa sigue cumpliendo su función original, que es la de devolver el valor por el que se multiplica la apuesta.

Ruleta

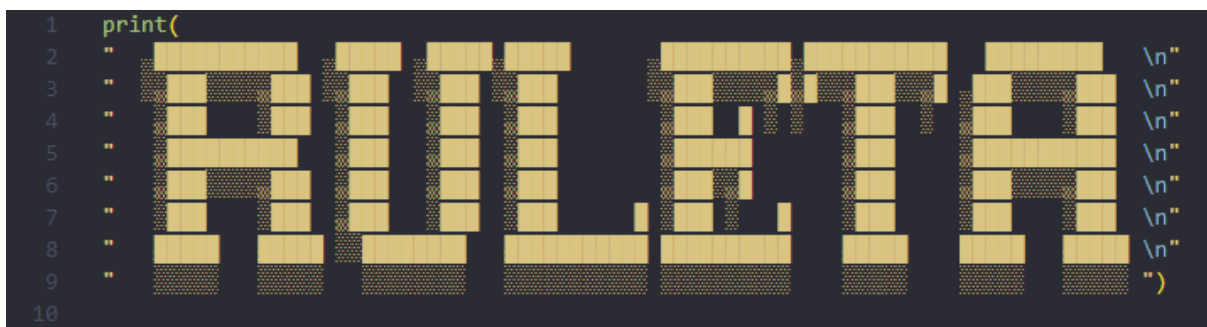
En cuanto al desarrollo de la ruleta comenzó siendo algo caótica ya que encontraba mil ejemplos en google en los cuales podía inspirarme, y hasta algunos con interfaz gráfica hechas en ventanas emergentes en los que la ruleta tenía una bola que giraba y se iluminaba el número ganador. Evidentemente a tal nivel de complejidad no aspiraba, entonces me inspiré en ideas más básicas que fui encontrando.

El primer problema llegó en el momento que ví dos ruletas que ambas me gustaba la forma en la que estaban hechas pero eran completamente distintas entre sí, por lo tanto no sacaba nada en claro y opté por comenzar la ruleta de cero con ideas propias mías.

Fase 1 - Idea y variables

El primer paso era encontrar una herramienta con la que pudiese escribir un letrero de "Ruleta", y por estúpido que parezca me gustaba la idea y a ello me puse.

```
1  print(  
2  "  
3  "  
4  "  
5  "  
6  "  
7  "  
8  "  
9  "  
10 ")
```



Nada más solucionar este detalle empecé a definir las variables y creé un menú para poder tomar distintas acciones en la ruleta, en el cual, dependiendo de la opción que eligieras se elegiría una de las funciones a ejecutar.

```
33 print("MENU: \n"  
34      "1. Apostar numero \n"  
35      "2. Apostar negro \n"  
36      "3. Apostar rojo \n"  
37      "4. Apostar verde \n"  
38      "5. Apostar pares \n"  
39      "6. Apostar impares \n"  
40      "7. Ver banco \n"  
41      "8. Dejar de apostar")  
42  
43 choice = int(input("Que desea hacer: "))  
44  
45  
46 bank = 1000  
47 print("Bank: ",bank)  
48 apuesta = int(input("Ingrese la cantidad que quiere apostar: "))  
49 ##numero_apostado = int(input("Ingrese el numero al que quiere apostar: "))
```

Fase 2 - Funciones y ejecución principal

Las funciones son un apartado que en si no han dado muchos problemas, ya que tenía muy claro que cada una iba a ser un tipo de apuesta (número en concreto, color y paridad), y todos los errores que iban surgiendo en ellas mientras avanzaba los corregía según más me convenía para que la ejecución fuese lo más limpia posible. Esto me llevó a perder algo de tiempo corriendo el programa numerables veces para poder sacar la mayor cantidad de errores posibles.

```
53 def apuesta_num():
54     global bank
55     numero_ganador = random.choice(lista_numeros)
56     print(x)
57     print("El numero premiado es:",numero_ganador)
58     tiradas + 1
59     if numero_ganador == 0:
60         print("El numero ganador es el 00 !!")
61         if numero_apostado == 0:
62             bank = bank + apuesta * 12
63             print("Bank: ",bank)
64         elif numero_apostado != 0:
65             bank = bank - apuesta
66             print("Bank: ",bank)
67
68     elif numero_apostado == numero_ganador:
69         print("Tu numero ha salido premiado !")
70         bank = bank + apuesta * 5
71         print("Bank: ", bank)
72     else:
73         bank = bank - apuesta
74         print("Tu numero no ha salido premiado !")
75         print("Bank: ", bank)
76
```

Esta es una de las funciones, en concreto es la situación en la que el jugador apuesta a un número en concreto. En primer lugar en todas las funciones importo la variable “bank” mediante un global, para poder jugar de esta forma con el dinero y posteriormente seleccionar un número ganador aleatorio de la lista principal de números.

Una vez elegido el número ganador la función comprueba varias cosas: en primer lugar revisa si el número ganador ha sido el 0 y si lo ha sido posteriormente revisa si el número que el jugador introdujo por consola coincide con el 0, en caso afirmativo multiplica la cantidad apostada * 12 y se lo añade a la variable bank que almacena el saldo total, si pierde la apuesta simplemente pierde la cantidad apostada. Si el número ganador no es el 0

simplemente revisa si el número ganador coincide con el apostado, si el jugador gana la apuesta multiplica la apuesta * 5.

El resto de funciones siguen una lógica similar, en el caso de apostar a color se comprueba si el número está en la lista de números de su color correspondiente y en cuanto a la paridad bastaba con saber el resto del número ganador para comprobar su paridad.

La ejecución principal tampoco tenía ninguna complejidad, un bucle *while* que le preguntara al usuario que quería hacer cada vez que una apuesta era cerrada. El usuario puede elegir cualquier tipo de apuesta (que funciona ejecutar) y después el programa preguntará al usuario cantidad a apostar y en caso necesario a que número en concreto quería apostar. También

hay una opción de revisar cuanto saldo queda para jugar, la cual creo que es algo inútil por que cada operación que se realiza después te muestra el saldo, pero bueno, la opción ahí está, y por último en caso de que el saldo sea igual a 0 se cerrará el programa y no dejará seguir jugando. También si en el menú se elige cualquier opción que no esté comprendida entre el 1 y el 7 el programa se cerrará ya que es como si no se hubiese elegido qué hacer. Bastante básico.

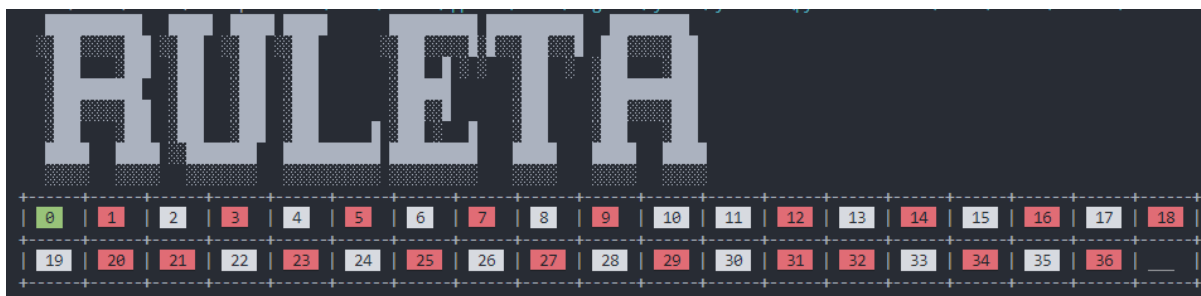
```
163 while choice < 8:
164     if choice == 1:
165         numero_apostado = int(input("Ingrese el numero al que quiere apostar: "))
166         apuesta_num()
167         choice = int(input("Que desea hacer: "))
168         apuesta = int(input("Ingrese la cantidad que quiere apostar: "))
169
170     if choice == 2:
171         apuesta_negro()
172         choice = int(input("Que desea hacer: "))
173         apuesta = int(input("Ingrese la cantidad que quiere apostar: "))
174
175     if choice == 3:
176         apuesta_rojo()
177         choice = int(input("Que desea hacer: "))
178         apuesta = int(input("Ingrese la cantidad que quiere apostar: "))
179
180     if choice == 4:
181         apuesta_verde()
182         choice = int(input("Que desea hacer: "))
183         apuesta = int(input("Ingrese la cantidad que quiere apostar: "))
184
185     if choice == 5:
186         #numero_apostado = int(input("Ingrese el numero al que quiere apostar: "))
187         apuesta_pares()
188         choice = int(input("Que desea hacer: "))
189         apuesta = int(input("Ingrese la cantidad que quiere apostar: "))
190
191     if choice == 6:
192         #numero_apostado = int(input("Ingrese el numero al que quiere apostar: "))
193         apuesta_impares()
194         choice = int(input("Que desea hacer: "))
195         apuesta = int(input("Ingrese la cantidad que quiere apostar: "))
196
197     if choice == 7:
198         print("Bank: ",bank)
199         choice = int(input("Que desea hacer: "))
200
201     if bank <= 0:
202         print("Se te acabó el dinero")
203         break
```

Fase 4 - PrettyTable

Lo último que quedaba del programa era darle una forma mínimamente visual, ya que no me veía capaz de realizar una ruleta que girase y fuese redonda y tampoco me parecía original copiar una de internet, lo cual podría haber hecho porque había muchas (de las cuales desconoce los métodos utilizados) por tanto decidí hacerlo mediante la única librería de tablas que conocía: tabulate.

Tabulate me parecía la opción perfecta, pero aquí comenzaron los problemas. Mi ordenador sobremesa con el que trabajaba no tenía pip instalado, lo cual era un problema ya de base. Instale una versión que no correspondía, pero antes de darme cuenta al ir a instalar Tabulate esté no respondía y python no importaba la librería. Luego de pasar bastante rato sin entender por qué no me importaba la librería hasta que me di cuenta de que la versión de pip era errónea y eso podría dar problemas. Actualicé la versión y para mi sorpresa Tabulate seguía sin funcionar como debería.

Me dí completamente por vencido y recurrí a otra librería, a mi parecer menos conocida o que al menos no había ni oído hablar de ella que es PrettyTable. No quise indagar ni complicarme, llevaba mas de dos horas buscando el error tonto por el cual no pudiese usar Tabulate en mi ordenador de sobremesa y no quería complicarme y tire por la tabla mas basica. Copié todos los codigos de color que me interesaban para darle el color al texto y al fondo que yo queria (de esta web: <https://stackoverflow.com/questions/287871/how-do-i-print-colored-text-to-the-terminal-in-python>) de esta forma la table se veía algo mas coloridad, similar a la que aparece en los casinos a un lateral de la ruleta que gira. El resultado fue este.



Al acabar todo esto lo único que me quedaba era repasar errores, revisar función a función si tenían algún fallo de ejecución, encontrarlo y ver que había escrito mal y simplemente ir poco a poco y con calma probando.

Bingo

El bingo podría decirse que es uno de los juegos más fáciles de este trabajo, pero a la vez digno y un reto a superar para mi persona.

Primero de todo me informé del funcionamiento del bingo, porque no soy un señor mayor que va a la residencia de ancianos a jugar al bingo todos los días, para entenderlo bien y posteriormente codificarlo. La idea principal es un cartón de bingo clásico de 5x5, es decir, 25 casillas con 3 botones, salir, dibujar un número y eliminar el número del cartón.

Primero no sabía si hacerlo como todo el grupo, es decir, de forma gráfica en el terminal o por el contrario usar algo diferente. Entonces, me acordé que un compañero de clase comentó sobre un módulo que servía para que se pudieran visualizar ventanas emergentes, entonces empecé a investigar. Antes de nada, decir que los módulos utilizados han sido:

random, os, tkinter, PIL y time.

Ese mundo tan inmenso es Tkinter, que en un principio parecía fácil de entender y programar pero cada vez que pones más código más complicado se pone y un mar de dudas empieza a llegar. Empecé aprendiendo lo básico, que era cómo podía abrir una ventana. (nombre_ventana)=Tk(), donde todo inicia. Desde ahí todo fueron avances, empecé a hacer las definiciones para el funcionamiento principal del código.

Aquí hice una lista con los 100 primeros números para luego, con un random el cual será usado para el botón que mostrará un número aleatorio con el que poder ir eliminando los dígitos del cartón..

```
num_elegidos=[]
numeros=[]
for i in range(1,101):
    numeros.append(i)

def numeros_random():
    global c
    c=[]
    c=random.sample(numeros,1)
    numeros.remove(c[0])
    text1.delete('1.0',END)
    text1.insert(INSERT,c[0])
```


Hasta aquí bien, pero ¿cómo hacer que, si sale un número del tablero del bingo, se pueda identificar del resto? Fácil, otra definición que le cambie el color del fondo a los “Label”, que es donde se guardan y muestran los números random del cartón. Así hasta que defina las 25 casillas. Siento que se puede mejorar esto con algún bucle, pero tras error y error no me salió. Luego la última definición que, básicamente servirá para poder mostrar los números aleatorios en el cartón.

```
def elim_nums():
    try:
        for i in range(len(listo)):
            if listo[i]==c[0]:
                text1.delete('1.0',END)
                if str(lab1.cget("text"))==str(c[0]):
                    lab1.config(bg="red")
                elif str(lab2.cget("text"))==str(c[0]):
                    lab2.config(bg="red")
                elif str(lab3.cget("text"))==str(c[0]):
                    lab3.config(bg="red")
```

```
def elegir_num():
    listo=[]
    while (len(listo)<=25):
        k=random.choice(range(1,101))
        if k not in listo:listo.append(k)
    return listo
```

Luego lo más importante era poner esos números en la ventana de Tkinter. ¿Cómo? Así:

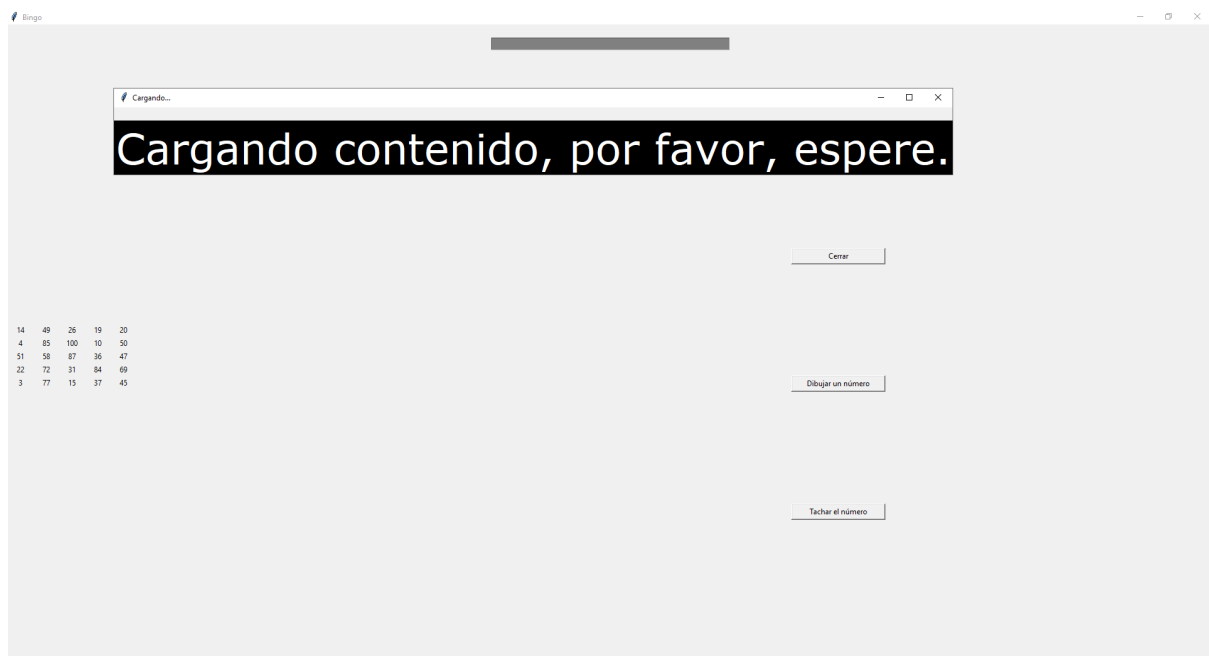
```
can1=Canvas(ventana,width=100,height=70)
lab1=Label(can1, text=listo[0],width=5,borderwidth=4, relief="solid",font=('Bold',80))
lab1.pack()
lab2=Label(can1, text=listo[1],width=5,borderwidth=4, relief="solid",font=('Bold',80))
lab2.pack()
lab3=Label(can1, text=listo[2],width=5,borderwidth=4, relief="solid",font=('Bold',80))
lab3.pack()
lab4=Label(can1, text=listo[3],width=5,borderwidth=4, relief="solid",font=('Bold',80))
lab4.pack()
lab5=Label(can1, text=listo[4],width=5,borderwidth=4, relief="solid",font=('Bold',80))
lab5.pack()
can1.pack(side=LEFT)
```

Crear un espacio para la fila de números (Canvas) y vas haciendo “Labels” para que aparezcan, además de invocarlos. Así de veces las filas que te hagan falta. Sobre los estilos hablaré más adelante.

Esta parte de código crea los botones en la ventana, dándole las funciones de las anteriores definiciones. Jugar con las coordenadas es todo un quebradero de cabeza.

```
bot1=Button(ventana,text='Cerrar',height=5,width=15,fg="red",bg="black",
activebackground="black",font=('Helvetica bold',20),command=ventana.quit)
bot1.place(relx = 0.850, rely = 0.25)
bot2=Button(ventana,text='Dibujar un número',height=5,width=15,fg="red",bg="black",
activebackground="black",font=('Helvetica bold',20),command=numeros_random)
bot2.place(relx = 0.850, rely = 0.45)
bot3=Button(ventana,text='Tachar el número',height=5,width=15,fg="red",bg="black",
activebackground="black",font=('Helvetica bold',20),command=elim_nums)
bot3.place(relx = 0.850, rely = 0.65)
```

Después de tener un código funcional, empecé a mejorar la estética. Al principio era bastante feo, no tenía coherencia ni orden.



Y, después de una tarde buscando y buscando y preguntando a google y al querido stackoverflow, encontré atributos bastante interesantes para hacerlo más bonito, siendo así la última versión:

91	9	45	67	75	Cerrar
86	68	16	49	6	
24	62	97	17	78	Dibujar un número
79	56	95	90	82	Tachar el número
99	63	22	47	26	

Además, antes de esta ventana, hay una que da la impresión de que los números se están cargando.

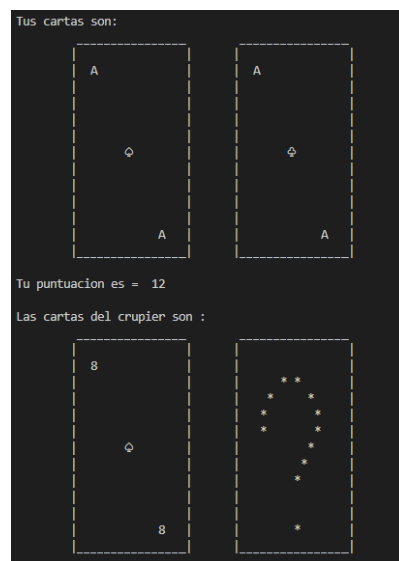
BlackJack

Mi parte del trabajo en el casino corresponde al Blackjack. En un primer momento pensé que sería fácil, ya que al fin y al cabo es un juego que conocía y pensaba que no sería difícil de programar.

Empecé programando poco a poco pero me di cuenta de que esta metodología no me servía, ya que al día siguiente sentía que había perdido todo lo que tenía, así que un domingo por la tarde me puse frente al ordenador y empecé a hacer el programa de 0 con diccionarios. En esta primera instancia me pareció algo fácil y asequible. El problema vino cuando le presenté lo que tenía a mi compañero Alex. Ahí es cuando vi todos mis errores, tan importantes y graves como que no tenía interfaz gráfica(eso sabía que no lo tenía pero no sabía cómo ponerle solución), no hice que el as pudiera valer 1 u 11, tampoco se podía ver los números que tenía el crupier, algo estúpido, ya que es imposible apostar sin saber lo que tiene el crupier.

```
if len(player_cards) == 2:
    if player_cards[0].card_value == 11 and player_cards[1].card_value == 11:
        player_cards[0].card_value = 1
        player_score -= 10
```

En ese momento me vine mucho abajo, ya que tendría que modificar el programa casi desde 0 porque aparte de todos esos errores faltaban muchas otras cosas. Fue ahí cuando empecé a investigar en internet y en foros viendo cómo lo hacía la gente, y me di cuenta de que la forma más sencilla es con una clase de carta y después ir creando variables para todo lo que necesite. La interfaz gráfica de las cartas la obtuve de internet, ya que por mi cuenta en ASCII quedaban mucho más feas y parecían a medio hacer. Además, me gustó mucho el detalle que tuvo el creador de poner al crupier la segunda carta con una interrogación y lo adherí a mi Blackjack. En cuanto al problema del as y su valor, intenté por muchas formas alcanzar una solución correcta y pedí ayuda a mi compañero Alex.



Otra de las cosas que he hecho que no corresponden a la realidad es que no sé hacer una inteligencia artificial para el crupier que elija cuándo pedir o no una carta, por lo que he hecho que cuando su puntuación sea 17 o menos (esto lo pone en una página sobre si apostar o no y cuándo hacerlo, esta página es una referencia ya que no encuentro la original: <https://apuestas.marathonbet.es/apostar-en-casino/guia-para-apostar-al-blackjack/>)

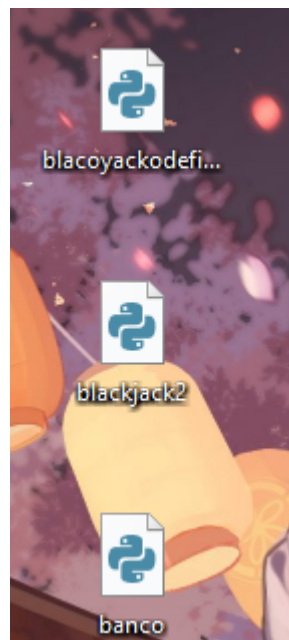
```
while crupier_puntuacion < 17:
    clear()

    print("EL CRUPIER DECIDE COGER OTRA CARTA.....")

    crupier_carta = random.choice(deck)
    crupier_cartas.append(crupier_carta)
    deck.remove(crupier_carta)
```

Para puntualizar y profundizar en el tema de las versiones del programa, he tenido un recorrido peculiar con estas, empecé con la versión inicial, que como he dicho antes

acabó en nada. Más tarde empecé con una segunda versión pero me di cuenta de que el nombre de las variables y el programa en sí mismo, por lo que fui modificando poco a poco tanto la estructura como las variables, y cuando fui a ejecutar, colapsaba por todas partes y no sabía como volver al inicio, así que decidí que con la estructura que tenía, empezar otro archivo desde 0 pensándolo todo mejor desde el principio, y en este sí que sí ya funcionó todo correctamente. En este último programa, como tenía miedo de que ocurriera lo mismo que lo que me pasó antes, cuando quería hacer un cambio importante hacía otro archivo Python y modificaba ahí, por lo que no había peligro a perderlo todo, aunque también podría haberlo hecho mediante git, en ese momento decidí hacerlo por lo fácil para mí en ese momento, pero si lo hubiera hecho con más tiempo y no dejándolo tan para el final, lo habría hecho por git.



Me sorprendió que al final el programa fuese tan largo, pero si lo pienso fríamente es normal, ya que hay muchísimas formas de acabar una partida y hay que recogerlas todas. Al final me ha gustado mucho el resultado, me parece muy interactivo gracias a la interfaz gráfica, que aunque es muy simple, es muy graciosa.

Por último, a última hora revisando el trabajo de mis compañeros me di cuenta de que me faltaba introducir la variable de créditos del banco para poder apostar, y su respectiva apuesta. Fue cuestión de un momento terminarlo, pero fue lo último que hice en el programa.

```
banco = 2000
jugador_cartas = []
crupier_cartas = []

jugador_puntuacion = 0
crupier_puntuacion = 0

clear()
apuesta = int(input("Cuanto quieres apostar"))
banco -= apuesta
```

Rama cambios

En un momento del desarrollo, viendo cómo funcionaban los programas del resto del equipo decidí cambiar algunas cosas en otra rama para no sobrescribir su trabajo y que funcionase mejor la implementación del menú. Estas modificaciones incluyen cambios menores como incluir *try except* para recoger cualquier comportamiento incorrecto de los programas, y añadir la opción de EXIT para volver al menú al terminar la ejecución de cada programa. Así mismo, existe una rama backup donde guardé la rama main sin ningún tipo de modificación para poder tener la versión original de cada programa en caso de querer recuperarlos.

Gestión de incidencias

Uno de los “errores” que continúan en el proyecto es, que en el caso del bingo, al ejecutarlo y llamar su módulo desde el menú todo funciona perfectamente, pero al salir desde el botón de cerrar, Visual Studio nos devuelve al menú, tal y como debía suceder, pero la ventana de bingo continúa abierta y debe ser cerrada manualmente. Intenté solucionarlo pero, con la falta de tiempo y el desconocimiento del módulo tkinter, la razón por la que este programa no pare su ejecución solo la conoce el demiurgo (asumo).

Conclusiones

Finalizando esta documentación, exponemos las conclusiones a las que hemos llegado. En general, el programa funciona y cumple con su propósito. Con más tiempo se podrían haber implementado más cosas, etc., pero no es un mal trabajo. El mayor problema en este proyecto ha sido el tiempo y la gestión del mismo, que se podrá tener en cuenta para mejorar la metodología de trabajos futuros.

Bibliografía

n.d. Index page. Accessed May 23, 2022. <http://forums.famitracker.com/>.

Belchenko, Alexander. n.d. "How do I print colored text to the terminal?" Stack Overflow.

Accessed May 23, 2022.

<https://stackoverflow.com/questions/287871/how-do-i-print-colored-text-to-the-terminal-in-python>.

Cebollada, Sergio. n.d. "Guía para apostar al Blackjack en 10 pasos." Blog Marathonbet.

Accessed May 23, 2022.

<https://apuestas.marathonbet.es/apostar-en-casino/guia-para-apostar-al-blackjack/>.

"Create ASCII text banners online." n.d. Manytools. Accessed May 23, 2022.

<https://manytools.org/hacker-tools/ascii-banner/>.

Anexo