¿Quién Quiere Ser Millonario?

Por Elian Ávila Osorio

¿Como se hizo?

El programa tiene 6 partes, divididas en 6 archivos:

- ***El simulador*** *(backend.py)*
  - ***La clase ‘Person’*** *(person.py)*
- ***El renderizador*** *(ui.py)*
- ***La interfaz principal*** *(main.py)*
- ***La base de datos*** *(util.py)*
- ***El creador de ‘dummies’**** *(dummygen.py)*

** El creador de ‘dummies’ no tiene una página de documentación aquí. Lo siento.*

El Simulador

El simulador se encarga de simular los juegos, las personas, y la cantidad de dinero dada cada día. Sus funciones son:

- *Simular las clasificatorias*
- *Simular el juego*
- *Tener en cuenta la cantidad de dinero dado.*

Variables

El simulador tiene un total de 7 variables globales.

***prizes**: Es una lista que contiene los premios para cada pregunta.*

***moneyGivenOut**: La cantidad de dinero que se ha premiado en total a cada concursante.*

***peopleToPass**: El número de personas que van a pasar a competir.*

***totalDaysPassed**: El número de días simulados hasta el momento.*

***peopleToCompete**: Lista que contiene las tuplas con las personas elegidas para cada día simulado.*

***peopleThatCalled**: Lista temporal usada para crear las personas que van a competir en las clasificatorias.*

***peopleThatPassed**: Lista temporal usada para guardar las personas que ganaron las clasificatorias. Esta se usa para crear la lista de tuplas **peopleToCompete**.*

La clase *Person.*

Cada persona es simulada con una clase llamada *Person.* Esta contiene:

Información personal:

- Nombre, NUIP y Edad.

Información del juego:

- Cuantos comodines se han usado, la integridad de la persona, la cantidad de dinero que tiene la persona actualmente, la última pregunta que contestó, cuanto tiempo tardó en contestar la última pregunta, el último día que participó, y los comodines que tiene actualmente.*

Información adicional:

- La ventaja que le otorga cualquier comodín que tenga actualmente.

** La integridad de la persona se usa al momento de decidir si esta va a usar un comodín. Si su integridad es baja, tiene mayor chance de usar un comodín. También, si su integridad es alta, el participante tiene mayor chance de salir voluntariamente cuando tenga lo suficiente.*

La clase *Person.*

La función de **Person**,

answer(),
(responder)

es la que se encarga de responder a las preguntas del juego. Primero calcula las probabilidades de elegir un comodín. Estas se basan en la dificultad de la pregunta, la integridad de la persona, y si la persona si quiera tiene comodines.

Después, calcula la probabilidad de responder correctamente, el tiempo que le tomo en responder, el comodín que uso, si va a usar la respuesta de los comodines y si va a retirarse después de esta pregunta.*

```python
def answer(self, diff):
        # CALCULATE WILDCARD PROBABILITY #
        wid = None

        if rng.randrange(15) > diff and len(self.wildcards) != 0 and rng.randint(0, 100) > self.integrity:
            wid = rng.choice(self.wildcards)
            self.wildcards.remove(wid)
        self.wildcard(wid)

        ans = {
            "correct":  diff * 4 * self.wildcardProb < rng.randint(0,100),
            "time":     diff * 3 + rng.randint(10,25),
            "wildcard": wid,
            "wildcardResponse": int(rng.randrange(15) > diff and rng.randint(0, 100) > self.integrity),
            "optout": rng.randrange(15) > diff and rng.randint(0, 100) < self.integrity and diff > 5
        }

        self.timeClass = ans["time"]

        return ans
```

** En los comodines “ayuda del público” y “llamada a un amigo”, se pregunta a alguien que respuesta cree que es correcta. El participante después tiene la opción de usar esta respuesta o no. Esto es lo que **wildcardResponse** almacena, un booleano que decide si el concursante usará la respuesta de estos comodines.*

Como simula las clasificatorias:

La función

classificationStep()
(paso de clasificatoria)

se encarga de simular las clasificatorias.

```
def classificationStep():
    resetPeopleThatCalled()

    pit = tuple(zip(peopleThatCalled[0], peopleThatCalled[1]))

    u.log(ui.title("[CLASSIFICATION ROUNDS START!]"))

    u.log(f"[START CLASSIFICATION ROUNDS: A TOTAL OF {peopleToPass*2} PEOPLE ARE COMPETING.]\n")

    for i in pit:
        u.log(i[0].name + "\n    vs.   \n" + i[1].name + "")
        u.log(ui.dlg("Host", f'{i[0].name}, {i[1].name}, {rng.choice(u.classificationQuestions)}') ,"transcript")
        if i[0].timeClass > i[1].timeClass:
            peopleThatPassed.append(i[1])
            u.log("\t - "+f'{i[1].name} beat {i[0].name} in answering the classification question by {i[0].timeClass-i[1].
        elif i[0].timeClass < i[1].timeClass:
            peopleThatPassed.append(i[0])
            u.log("\t - "+f'{i[0].name} beat {i[1].name} in answering the classification question by {i[1].timeClass-i[0].
        else: # CHOOSE AT RANDOM IF BOTH ANSWERED AT THE SAME TIME.
            personThatWon = rng.choice([i[0], i[1]])
            u.log("\t - "+f"Both people answered at the same time, so {personThatWon.name} was chosen randomly.")
            peopleThatPassed.append(personThatWon)
        u.log("\n","transcript")
    u.log(f"\n[END CLASSIFICATION ROUNDS: {peopleToPass} PEOPLE HAVE ENTERED THE COMPETITION.]\n")
    u.log("[PEOPLE THAT WERE SELECTED:]")
    for i in peopleThatPassed:
        u.log(str(i))
    pairings = tuple(zip(peopleThatPassed[:math.ceil(peopleToPass/2)], peopleThatPassed[math.ceil(peopleToPass/2):]))
    u.log("[COMPETITION PAIRINGS:]")
    for i in pairings:
        u.log("\t"+f' - ({i[0].name} vs. {i[1].name})' + "\n")
        peopleToCompete.append(i)
```

Como simula las clasificatorias:

Primero, se reinicia la lista de las personas que van a estar en las clasificatorias, ya que esta se crea dinámicamente, y los parámetros se pueden cambiar.

Después, esta parte se encarga de emparejar las 2 partes de ***peopleThatCalled*** en una sola lista de tuplas.

```
def resetPeopleThatCalled():
    global peopleThatCalled
    peopleThatCalled = [dummygen.genPeople(peopleToPass), dummygen.genPeople(peopleToPass)]

def classificationStep():
    resetPeopleThatCalled()

    pit = tuple(zip(peopleThatCalled[0], peopleThatCalled[1]))

    u.log(ui.title("[CLASSIFICATION ROUNDS START!]"))

    u.log(f"[START CLASSIFICATION ROUNDS: A TOTAL OF {peopleToPass*2} PEOPLE ARE COMPETING.]\n"

    for i in pit:
```

Como simula las clasificatorias:

Cada objeto **Person** tiene una variable llamada *timeClass*. Este se usa para guardar el tiempo que le toma a un participante el responder a una pregunta. Para las clasificatorias, se simula con la siguiente línea de código:

```
random.randint(30, 60)
```

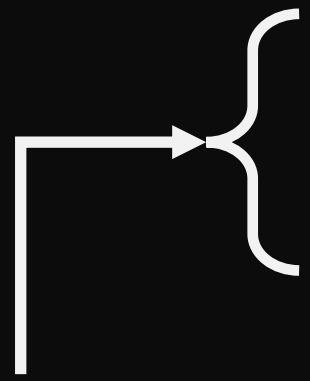
Es un valor entre 30 y 60 segundos. Este es aleatorio para cada persona que se genere. Esta parte de la función compara las personas en cada tupla de ***pit***, y si una persona tardó en responder la pregunta, la elimina.

```
u.log(ui.title("[CLASSIFICATION ROUNDS START!]"))

u.log(f"[START CLASSIFICATION ROUNDS: A TOTAL OF {peopleToPass*2} P

for i in pit:
    u.log(i[0].name + "\n    vs.   \n" + i[1].name + "")
    u.log(ui.dlg("Host", f'{i[0].name}, {i[1].name}, {rng.choice(u.
    if i[0].timeClass > i[1].timeClass:
        peopleThatPassed.append(i[1])
        u.log("\t - "+f'{i[1].name} beat {i[0].name} in answering t
    elif i[0].timeClass < i[1].timeClass:
        peopleThatPassed.append(i[0])
        u.log("\t - "+f'{i[0].name} beat {i[1].name} in answering t
    else: # CHOOSE AT RANDOM IF BOTH ANSWERED AT THE SAME TIME.
        personThatWon = rng.choice([i[0], i[1]])
        u.log("\t - "+f"Both people answered at the same time, so t
        peopleThatPassed.append(personThatWon)
    u.log("\n","transcript")
u.log(f"\n[END CLASSIFICATION ROUNDS: {peopleToPass} PEOPLE HAVE E
```

Como simula las clasificatorias:



```
            personThatWon = rng.choice([i[0], i[1]])
            u.log("\t - "+f"Both people answered at the same time, so {personThatWon.name} was chosen randomly.")
            peopleThatPassed.append(personThatWon)
    u.log("\n","transcript")
u.log(f"\n[END CLASSIFICATION ROUNDS: {peopleToPass} PEOPLE HAVE ENTERED THE COMPETITION.]\n")
u.log("[PEOPLE THAT WERE SELECTED:]")
for i in peopleThatPassed:
    u.log(str(i))
pairings = tuple(zip(peopleThatPassed[:math.ceil(peopleToPass/2)], peopleThatPassed[math.ceil(peopleToPass/2):]))
u.log("[COMPETITION PAIRINGS:]")
for i in pairings:
    u.log("\t"+f' - ({i[0].name} vs. {i[1].name})' + "\n")
    peopleToCompete.append(i)
```

Aquí es donde se empaquetan los pares de personas que van a concursar cada día.

Esto pasa por cada persona que ganó las clasificatorias y los junta con otra persona.

Después, pasa esta información a la lista ***peopleToCompete***.

Como simula el juego:

Aunque técnicamente la función *simulateDay()* se encarga de simular cada día de juego, la función

askQuestions()
(haz preguntas)

*se encarga de simular la **lógica** del juego completo en **96 líneas de código.***

```
def askQuestions(personPair):
    questionsAsked = 0
    currentSafeguard = 0
    currentContestant = 0
    sets = [rng.randint(0,46), rng.randint(0,46)]
    moneyGiven = 0
    hasLost = False
    curcon = personPair[currentContestant]

    curcon.wildcards = [0,1,2]

    def changeSafeguard():
        u.log(ui.dlg("Host",f"We've now passed a safeguard, you can now lose any question, and you'll
still win {formattedMoney}."))
        nonlocal currentSafeguard
        currentSafeguard = prizes[questionsAsked]

    while questionsAsked < 14:
        u.log("\n")
        if currentContestant == 0:
            u.log(ui.foot("FIRST CONTESTANT'S TURN:"))
        else:
            u.log(ui.foot("SECOND CONTESTANT'S TURN:"))

        while hasLost is False:
            formattedMoney = ui.format(prizes[questionsAsked] * 100000, "money")

            u.log(ui.title(f"\n\t# -- -- -- QUESTION #{questionsAsked+1} -- -- -- #\n"))
            u.log(ui.dlg("Host", f"{curcon.name},
{u.questions[questionsAsked+sets[currentContestant]]}"))

            personAnswer = curcon.answer(questionsAsked)
            curcon.lastQuestion = [questionsAsked, u.questions[questionsAsked+sets[currentContestant]
            curcon.lastDay = totalDaysPassed

            if personAnswer["wildcard"] is not None:
                wildcardToPlay = ['50/50', 'Public Opinion', 'Lifeline'][personAnswer['wildcard']]
                wildcardResponse = ["Hmmm... I think I will not.", "I think I will."]
                u.log(ui.dlg("Host",f"It seems like our contestant has decided to use a wildcard."))
                u.log(ui.dlg(curcon.name,f"I want to play the {wildcardToPlay} wildcard."))
                u.log(ui.dlg("Host", f'Alright, the {wildcardToPlay} wildcard.'))
                if personAnswer['wildcard'] == 0:
                    u.log(ui.dlg("Host",f'Two options have been removed.'))
                elif personAnswer['wildcard'] == 1:
                    u.log(ui.dlg("Host",f'The public is weighing in... It looks like they want answe
#{rng.randint(1,4)}! Will you choose that one?'))
                u.log(ui.dlg(curcon.name, wildcardResponse[personAnswer["wildcardResponse"]]))
                u.log(ui.dlg("Host", "Alright then! Choose your answer when you're ready."))
```

Como simula el juego:

Aquí es donde se inicializa el juego.

Se reinicia el número de preguntas ya preguntadas *(questionsAsked)*, el seguro que tiene el jugador *(currentSafeguard)*, el estado del jugador *(hasLost)*, así como información adicional como la cantidad dada en esa sesión *(moneyGiven)*, los conjuntos de preguntas que cada participante tendrá *(sets)* y el participante actual *(currentContestant)*

La variable ***curcon*** se utiliza para cambiar la información del participante en sí. Aquí también se reinician los comodines del participante.

```
def askQuestions(personPair):
    questionsAsked = 0
    currentSafeguard = 0
    currentContestant = 0
    sets = [rng.randint(0,46), rng.randint(0,46)]
    moneyGiven = [0,0]
    hasLost = False
    curcon = personPair[currentContestant]

    curcon.wildcards = [0,1,2]

    def changeSafeguard():
        u.log(ui.dlg("Host",f"We've now passed a saf
```

Como simula el juego:

La subfunción

changeSafeguard()
(cambiar seguro)

se encarga de cambiar el seguro del jugador, porque el bucle en el que la lógica se ocurre no permite cambiar a ***currentSafeguard*** *por motivos de alcance de variables.*

```
def changeSafeguard():
    u.log(ui.dlg("Host",f"We've now passed a s
    nonlocal currentSafeguard
    currentSafeguard = prizes[questionsAsked]
```

Como simula el juego:

El bucle *while*

questionsAsked < 14
(preguntas preguntadas son menores que 14)

se encarga de introducir el participante, y también de transferir el dinero ganado a la cuenta de banco del participante al final de todo.

La variable **money** *del participante es volátil, ya que puede reiniciarse cada vez que pierda el juego. Si un concursante pierde más de una vez, el dinero que ya ganó no cambia.*

```
while questionsAsked < 14:
        u.log("\n")
        if currentContestant == 0:
            u.log(ui.foot("FIRST CONTESTANT'S TURN:"))
        else:
            u.log(ui.foot("SECOND CONTESTANT'S TURN:"))

        while hasLost is False:


          […]


        formattedMoney = ui.format(curcon.money * 100000, "money")

        if currentContestant == 0:
            u.log(ui.dlg("Host", f"And with that, {curcon.name} leaves with a
            moneyGiven[0] = curcon.money
            curcon.transferToBank()
            hasLost = False
        else:
            u.log(ui.dlg("Host", f"And with that, {curcon.name} leaves with a
            moneyGiven[1] = curcon.money
            curcon.transferToBank()
            break
```

Como simula el juego:

El bucle *while*

hasLost is False
(ha perdido es falso)

es el que se encarga de la lógica principal. Esta primera parte se encarga de presentar la suma de dinero en una manera más legible, preguntar una pregunta al participante.

personAnswer *corre la función* ***answer()****, que cada* ***Person*** *tiene.*

```
while hasLost is False:
    formattedMoney = ui.format(prizes[questions

    u.log(ui.title(f"\n\t# -- -- -- QUESTION #{
    u.log(ui.dlg("Host", f"{curcon.name}, {u.qu

    personAnswer = curcon.answer(questionsAsked
    curcon.lastQuestion = [questionsAsked, u.qu
    curcon.lastDay = totalDaysPassed

    if personAnswer["wildcard"] is not None:
```

Como simula el juego:

Después de preguntar la pregunta,

if personAnswer[“wildcard”] is not None
(si la respuesta de la persona tiene un comodín)

*es el que se encarga de ver si la persona usó un comodín. Si lo usó, la función **answer()** del participante ya ajustó la probabilidad de responder correctamente al momento de preguntar.*

*Esta proposición “if” **solamente es para escribir el juego en el archivo ‘transcript.txt’, y no tiene ningún otro propósito.***

```
if personAnswer["wildcard"] is not None:
    wildcardToPlay = ['50/50', 'Public Opini
    wildcardResponse = ["Hmmm... I think I w
    u.log(ui.dlg("Host",f"It seems like our
    u.log(ui.dlg(curcon.name,f"I want to pla
    u.log(ui.dlg("Host", f'Alright, the {wil
    if personAnswer['wildcard'] == 0:
        u.log(ui.dlg("Host",f'Two options ha
    elif personAnswer['wildcard'] == 1:
        u.log(ui.dlg("Host",f'The public is
        u.log(ui.dlg(curcon.name, wildcardRe
        u.log(ui.dlg("Host", "Alright then!
    else:
        u.log(ui.dlg("Host",f'You can now ca
        u.log(ui.italic(f'\t - {curcon.name}
        u.log(ui.dlg("Host",f'Alright! Will
        u.log(ui.dlg(curcon.name, wildcardRe
        u.log(ui.dlg("Host", "Alright then!
```

Como simula el juego:

Después de ver si se usó un comodín,

if personAnswer[“correct”]
(si la respuesta de la persona es correcta)

es el que se encarga de ver si la respuesta de la persona es correcta. Esta ya se calculó en ***answer()****.*

Si la respuesta es correcta, el programa le da la cantidad correspondiente de dinero a la persona.[1]

Si pasó por un seguro, el programa corre la subfunción ***changeSafeguard()****.*[2]

Y por último, si el participante quiere retirarse, el programa lo deja irse con el dinero que ya ha ganado.[3]

```
if personAnswer["correct"]:
    u.log(ui.italic(f"\t - {curcon.name} answers c
    u.log(ui.dlg("Host",f'That is correct, well do
    if questionsAsked == 4 or questionsAsked == 9:
        changeSafeguard()
    curcon.money = prizes[questionsAsked]
    if personAnswer["optout"]:
        u.log(ui.dlg("Host", f"Huh? It appears our
        u.log(ui.dlg(curcon.name, "Yes."))
        u.log(ui.dlg("Host", f"Alright then! If th
        break
    questionsAsked += 1
    if questionsAsked == 15:
        print(f"{curcon.name} has just won the gra
        break
```

Como simula el juego:

Después de esto, incrementa el número de preguntas respondidas, e inicia de nuevo.[4]

*Si ya no hay más preguntas que responder, el bucle **hasLost is False** termina, y el participante gana el premio final.*[5]

```
if personAnswer["correct"]:
    u.log(ui.italic(f"\t - {curcon.name} answers c
    u.log(ui.dlg("Host",f'That is correct, well do
    if questionsAsked == 4 or questionsAsked == 9:
        changeSafeguard()
    curcon.money = prizes[questionsAsked]
    if personAnswer["optout"]:
        u.log(ui.dlg("Host", f"Huh? It appears our
        u.log(ui.dlg(curcon.name, "Yes."))
        u.log(ui.dlg("Host", f"Alright then! If th
        break
questionsAsked += 1
if questionsAsked == 15:
    print(f"{curcon.name} has just won the gra
    break
```

Como simula el juego:

Si el participante responde de manera incorrecta, el dinero que había ganado se reduce al último seguro que tenía.[6]

El dinero dado hasta el momento se actualiza[7], y el participante pierde.[8]

```
else:
    u.log(ui.italic(f"\t - {curcon.name} answers incorr
    u.log(ui.dlg("Host",f"I'm afraid that is incorrect.
curcon.money = currentSafeguard
moneyGiven[currentContestant] = currentSafeguard
hasLost = True
```

Como simula el juego:

Al final de cada sesión, es decir, al final de ***askQuestions()***, se escribe el dinero dado a cada participante en el archivo *transcript.txt*.

Después de esto, la cantidad de dinero dado al final de la sesión es añadida al contador total.

```
u.log(f"[MONEY GIVEN OUT ON DAY {totalDaysPassed+
u.log("\t" + f'{personPair[0].name} received {ui.
u.log("\t" + f'{personPair[1].name} received {ui.
u.log("\t" + f'In total, the program gave {ui.for

global moneyGivenOut
moneyGivenOut += moneyGiven[0] + moneyGiven[1]
```

Como simula un día:

Cada día se simula con la función ***simulateDay()***.

Simplemente elige el par que va a competir ese día basado en el número de días que han pasado, y los pone en ***askQuestions()***.

Después de que termine el programa, la función avanza el contador de los días.

```
def simulateDay():
    global totalDaysPassed
    competingPair = peopleToCompete[totalDaysPassed]

    u.log(ui.title(f'\n\n\n\nDAY #{totalDaysPassed+1} -
    askQuestions(competingPair)
    totalDaysPassed += 1
```

Como simula todas las personas:

Para simular cada una de las personas en memoria, el programa usa la función

simulateDuration().
(Simular duración)

Esta desactiva la función de estilización ANSI del módulo *ui.py**, reinicia el archivo *transcript.txt*, corre las clasificatorias, y por cada par de personas en ***peopleToPass***, simula un día con ***simulateDay()***. Después de todo, escribe al archivo *transcript.txt* datos finales de la simulación, y reactiva la estilización ANSI.

```
def finalLog():
    u.log("\n### ---------------------------------------- ###\n")
    u.log("[FINAL LOG:]")
    u.log("\t[PARTICIPANT'S STATUS:]")
    for i in peopleThatPassed:
        u.log(i.logstr("\t\t"))
    u.log("\t[MONEY GIVEN OUT OVERALL:]")
    u.log("\t\tIn total, the program gave out " + ui.format(moneyGivenOut *

def simulateDuration():
    ui.ANSIdisable = True
    u.clearLog("transcript")
    classificationStep()
    for i in range(int(peopleToPass/2)):
        simulateDay()
    finalLog()
    ui.ANSIdisable = False
```

** Se tiene que desactivar la estilización ANSI, porque si no se hace, códigos de estilo ANSI se escribirían en el archivo **transcript.txt**, y los códigos ANSI solo se supone que se deban ver en consola.*

Como se renderiza el programa:

Para estilizar el programa, y también darle algo de UI para que el usuario pueda interactuar con el programa, escribí la librería ***ui.py***, para cualquier proyecto de Python que tuviera. Este usa códigos ANSI para estilizar la consola, y también tiene funciones para generar listas numéricas, crear *inputs*, diálogos, errores personalizables, entre otras cosas.

Como se usa el programa:

main.py se encarga de mostrar el menú principal. Este usa bucles y funciones de input para funcionar. Solamente interactúe con el teclado, escribiendo el numero de la opción que quiera seleccionar. El módulo ***ui.py*** hace la mayoría de la carga pesada en términos de limpiar la entrada del usuario, y asegurar de que no se entren valores inválidos.

Como el programa lee archivos:

util.py se encarga de leer todo. Tiene funciones para leer, escribir, y reiniciar archivos, así como una función de ***clamp()*** que no se encuentra en Python por default. Este módulo también tiene el arte de los créditos y las preguntas que se usan en la clasificatoria y en el juego, por lo que se le otorgó el nombre de *la base de datos*. En realidad, este archivo es reusado de otro proyecto de la universidad, y solo me interesaba la parte de leer los archivos. Siempre se cierran los escáneres cuando se usan, así que no hay que preocuparse acerca de archivos abiertos en el fondo siendo inutilizados.

Cosas que no pude implementar:

- El número de comodines que una persona ha usado se implementó, pero este no se muestra en la función __str__.

- Hay una función redundante en **Person** que hace lo mismo que __str__, pero peor. Es **logstr()**.

- Técnicamente se podría usar la información de las respuestas correctas para que el archivo *transcript.txt* no solamente tenga “Persona respondió correctamente/incorrectamente”, y que tenga lo que la persona respondió.

- Se tiene que desactivar la estilización ANSI antes de simular porque antes, el contenido de *transcript.txt* era solamente visible en la consola. Pero, cuando decidí poner el output de ***backend.py*** en *transcript.txt*, ya había usado un montón de funciones ANSI. Estas se podrían remover de ***backend.py*** y no causarían daño alguno.

- Música en los créditos :(

Gracias por ver.

Documentación hecha por Elian Avila Osorio,
para el proyecto final de Lenguajes de Programación.