

Proyecto Final ScrabbleAR

Julia Saenz

Seminario de Lenguajes Python 2020

Profesoras Claudia Banchoff
 Viviana Harari
JTP: Sofía Martín
 Juan Pablo Martínez
Ayudantes: Gaspar Rajoy
 Tomás Barbieri
 Federico Otarán
 Facundo Díaz Gira
Colaboradores: Nicolás Epíscopo.
 Ulises Cura Jauregui
 Agustín Genoves
 Mateo Durante
 Marianela Gauto
 Yanina Echevarría

Facultad de Informática
Universidad Nacional de La Plata

ÍNDICE

1. Introducción	1
2. Reglas de ScrabbleAR	1
3. Temas estudiados (marco teórico)	4
3.1. Sobre entornos virtuales (virtual env, conda, etc)	4
3.2. Sobre PySimpleGUI	5
3.3. Sobre Pattern	5
4. Problemas y soluciones surgidas sobre el desarrollo	6
4.1. Estructura del programa	6
4.1.1. Organización del código	7
4.2. La evaluación de palabras	7
4.3. La configuración de niveles	8
4.3.1. La ventana de configuración	8
4.4. Pausa y reanudación de la partida	8
4.5. Tableros generativos	9
5. Consideraciones éticas sobre el desarrollo	10
5.1. Accesibilidad del programa	10
5.2. Privacidad de la información	10
5.3. Licencias de los software de desarrollo	10
5.4. Licencia del software desarrollado	10
5.5. Documentación generada	11
6. Conclusiones y trabajos futuros	11
7. Referencias	13
8. Anexo A: Guía de Usuario	15
9. Anexo B: Guía para el desarrollador	21

1. INTRODUCCIÓN

En este informe se detalla el proceso de desarrollo del juego ScrabbleAR, realizado para la materia Seminario de Lenguajes Python de la Facultad de Informática de la UNLP en el primer cuatrimestre lectivo. La finalidad del proyecto fue el aprendizaje del lenguaje de programación Python y el uso de los módulos PySimpleGUI y Pattern.

2. REGLAS DE SCRABBLEAR

El juego se basa en las reglas del Scrabble tradicional: ambos jugadores deben utilizar letras, cada una con un puntaje, de una bolsa (en este caso un arreglo de fichas) para formar palabras y ubicarlas en un tablero con el fin de sumar puntos.

A cada jugador se le dan al azar 7 letras de la bolsa y se elige uno de los dos para comenzar la partida. En el primer turno una de las fichas debe ser colocada en el casillero del medio, y en cualquier otro turno las palabras pueden ser ubicadas en cualquier parte del tablero.

Las fichas pueden disponerse de forma horizontal o vertical y cuando se tenga la palabra se debe presionar el botón de “Terminar Turno”. Si la palabra ingresada es válida se sumarán los puntos acumulados y se pasará al turno del otro jugador; en caso de no serlo, se sacarán las fichas ingresadas del tablero y se le permitirá intentar otra combinación de fichas.

El tablero cuenta con casilleros especiales que cambian el puntaje de una ficha o palabra. Estos casilleros especiales son:

- Doble Letra: duplica el valor de la ficha en ese casillero
- Triple Letra: triplica el valor de la ficha en ese casillero
- Doble Palabra: duplica el valor de toda la palabra
- Triple Palabra: triplica el valor de toda la palabra
- Menos Uno: resta 1 punto a la ficha en ese casillero
- Menos Dos: resta 2 puntos a la ficha en ese casillero
- Menos Tres: resta 3 puntos a la ficha en ese casillero

El juego termina cuando se acaba el tiempo o cuando se terminan las fichas de la bolsa. Se restan los puntos correspondientes a las fichas que siguen en el atril y el ganador es aquel que haya acumulado más puntos. En caso de que el puntaje sea uno de los 10 mejores, tendrá la opción de guardarla en el “Top 10”.

En cualquier momento de la partida puede decidir “Guardar Partida” lo que cerrará y guardará esa partida, que luego puede ser retomada eligiendo “Continuar” en el inicio. Solo se guardará la última partida, por lo que si sobre-

escribe un juego, no podrá recuperarlo.

Las reglas expuestas hasta este punto fueron determinadas de antemano por la cátedra; las reglas expuestas de aquí en adelante son aquellas que decidí al principio o en el proceso de desarrollo del juego.

En el Inicio y a lo largo de la partida es posible configurar la dificultad del juego, ya sea con los niveles Fácil, Medio o Difícil predeterminados, o combinando su propio nivel customizado. El nivel varía:

- La dificultad de la Computadora
- La cantidad de fichas en la bolsa
- El puntaje de cada ficha
- El tablero
- El tiempo de juego
- Los tipos de palabras válidos

Cada nivel tiene esta configuración:

- Fácil

■ Dificultad computadora: Elige la mejor palabra de máximo 5 letras y la posiciona en un lugar aleatorio del tablero

- Cantidad de fichas:

- a, e: 12
- o: 9
- i, s: 6
- u, n, d, r: 5
- l, c: 4
- b, m, p, g, h: 2,
- f, v, j, k, ñ, q, t, w, x, y, z: 1

- Puntos fichas:

- a, e, i, o, u, s, n, l, r, t: 1
- d, g: 2
- c, m, b, p: 3
- f, h, v, y: 4
- j: 6
- k, ñ, z: 8
- q, w, x: 10

- Tablero:

- doble letra: 20
- triple letra: 11
- doble palabra: 16
- triple palabra: 4
- menos uno: 16
- menos dos: 8

- menos tres: 0
- Tiempo: 3 minutos
- Tipos de palabras admitidas: sustantivos, adjetivos y verbos
- Medio:
 - Dificultad computadora: elige la mejor palabra y la posiciona en un lugar aleatorio del tablero
 - Cantidad de fichas:
 - a, e: 11
 - o: 8
 - s: 7
 - i: 6
 - u, n, r: 5
 - d, l, c, t: 4
 - b, m, g, h,: 3
 - p, g, h: 2
 - f, v, j, k, ñ, q, t, w, x, y, z: 1
 - Puntos fichas:
 - a, e, i, o, u, s, n, l, r, t, c, d, g: 1
 - m, b, p: 2
 - f, h, v, y: 3
 - j: 5
 - k, ñ, z: 6
 - q, w, x: 8
 - Tablero:
 - doble letra: 20
 - triple letra: 16
 - doble palabra: 4
 - triple palabra: 4
 - menos uno: 16
 - menos dos: 8
 - menos tres: 8
 - Tiempo: 2 minutos
 - Tipos de palabras admitidas: adjetivos y verbos
- Difícil:
 - Dificultad computadora: elige la mejor palabra de máximo 5 letras y la posiciona en el lugar que le de más puntos
 - Cantidad de fichas:
 - a, e: 11
 - o: 8
 - s: 7
 - i, u: 6

- n, r: 5
- d, l, c, t: 4
- b, m: 3
- p, g, h: 2
- f, v, j, k, ñ, q, w, x, y, z: 1

■ Puntos fichas:

- a, e, i, o, u, s: 0
- n, l, r, t, c, d, g: 1
- m, b, p: 2
- f, h, v, y: 3
- j: 5
- k, ñ, z: 6
- q, w, x: 8

■ Tablero:

- doble letra: 16
- triple letra: 4
- doble palabra: 12
- triple palabra: 4
- menos uno: 12
- menos dos: 16
- menos tres: 16

■ Tiempo: 1 minuto

■ Tipos de palabras admitidas: adjetivos y verbos

3. TEMAS ESTUDIADOS (MARCO TEÓRICO)

3.1. Sobre entornos virtuales (virtual env, conda, etc)

Un entorno virtual es una herramienta que permite separar un proyecto específico junto a sus dependencias y librerías en un solo lugar de forma que no interfiera con las dependencias con otros proyectos¹.

Puede ocurrir que diferentes aplicaciones, por ejemplo, requieran diferentes versiones del mismo paquete (por ejemplo, PySimpleGUI) para funcionar correctamente pero, por defecto, los paquetes instalados mediante pip se guardan en el mismo directorio y Python no puede diferenciar entre distintas versiones de un mismo paquete, ya que tienen el mismo nombre. En esta situación, utilizando un entorno virtual, se crea un árbol de directorios (por defecto llamada venv) que contenga la versión de Python particular o cualquier paquete o librería para la cual se necesite una versión específica.

El paquete de Python para esta tarea se puede realizar fácilmente mediante la herramienta virtualenv. Venv, un módulo simplificado de virtualenv el cual

¹ (Agasthyan, 2017)

pierde algunas funcionalidades, está incluido con la versión estándar de Python 3, aunque también puede instalar virtualenv mediante pip.

Conda es un sistema de manejo de paquetes, dependencias y entornos de código abierto. Es, esencialmente, un reemplazo de virtualenv. La principal diferencia con este es que no está limitado a Python, sino que puede utilizarse con otros lenguajes y que también es posible instalar paquetes a través de conda (en lugar de hacerlo mediante pip).

■ 3.2. Sobre PySimpleGUI

PySimpleGUI es un framework para el desarrollo de interfaces gráficas (GUI) en Python lanzado en 2018 por la PySimpleGUI Organization que empaqueta elementos de otros paquetes (Tkinter, PyQt, wxPython y Remi) y los hace más fáciles de utilizar.

El objetivo de este paquete es facilitar el diseño y programación de interfaces gráficas, tanto para programadores principiantes como para experimentados. “El objetivo es tener tu GUI corriendo en cuestión de minutos, no horas ni días”² menciona la descripción del proyecto, y justamente es la simpleza de este paquete y su facilidad de uso la razón por la cual se destaca entre sus competidores. Se puede instalar a través del instalador de paquetes de Python: pip.

El paquete funciona haciendo uso de Elementos, una versión de PySimpleGUI de Widgets. Estos elementos consisten en textos, botones, popups, listas, y muchos más que se pueden ordenar en listas anidadas para crear el Layout, que se agrega al elemento Window para crear la ventana visible. Cada elemento cuenta con una serie de parámetros definidos por defecto que pueden modificarse, dándole la posibilidad al programador de controlar múltiples variables: color, tamaño, estado inicial, entre otros.

Otro factor importante es el uso de eventos y valores; los eventos refieren a los botones que se seleccionen dentro de una ventana y los valores a cualquier tipo de input por parte del usuario cómo ingresar texto o elegir una opción de una lista. Mediante la lectura de eventos y valores dentro de un loop, se maneja el programa. El loop es necesario para que el programa repita la lectura luego de la primera iteración.

■ 3.3. Sobre Pattern

Pattern es un módulo gratuito de minería web para Python que tiene herramientas para minería de datos, procesamiento de lenguaje natural, machine learning, análisis de red y visualización de canvas.³ Para este trabajo se utilizó únicamente el módulo pattern.es, que contiene un etiquetador de palabras, pudiendo identificar sustantivos, adjetivos, adverbios, verbos, etc en una frase, y herramientas para la conjugación, singu-

² PySimpleGUI, n.d. Texto original: “The goal is to get you running on your GUI within minutes, not hours nor days”.

³ Pattern Wiki, n.d.

larización y pluralización de verbos.

Una de las funciones más importantes del módulo es la función `parse()`, la cual recibe un `string` y devuelve un árbol de objetos anidados con las etiquetas correspondientes al tipo de palabra. Esta función se entrenó usando 1.5 millones de palabras etiquetadas de Wikicorpus y tiene una exactitud de 92%.

Hay también módulos para inglés (exactitud de 95% en el parser), alemán (exactitud de 85% en el parser), francés , italiano (exactitud de 92%) y holandés (exactitud de 91% en el parser). El módulo más desarrollado es definitivamente el inglés, ya que es el idioma original para el que se desarrolló, teniendo no solo más precisión, sino que también funciones como: superlativos, comparativos, cuantificación, sugerencias ortográficas, etc.

4. Problemas y soluciones surgidas sobre el desarrollo

4.1. Estructura del programa

El primer problema del proyecto que hubo que enfrentar fue el cómo encarar la estructura del programa. Se nos dio la opción desde la cátedra de utilizar o no objetos según nuestra preferencia, por lo que llegado el caso tuve que detenerme a pensar qué forma sería no sólo la más eficiente, sino la que sería más fácil de expandir y agregar funcionalidades.

La decisión final fue de utilizar las siguientes clases:

- Casillero
- Tablero
- Jugador
- Computadora
- Turno

Primero surgió la clase Casillero para manejar el valor de cada casilla y si tenía o no una letra; luego la clase Tablero, que maneja y ordena todas las instancias de Casillero, funcionando como intermedio entre cada casilla y los jugadores. Jugador fue una clase evidente para tener forma de manejar su nombre, su atril y su puntaje; y Computadora terminó siendo una extensión de la clase Jugador, ya que debía manejar muchas de las mismas variables y métodos, con algunas divergencias en, por ejemplo, el modo de crear una palabra. Por último, una clase que apareció por necesidad más adelante en el proceso fue la clase Turno, que contiene todas las variables que cambian en cada turno o partida como el último evento seleccionado, las palabras totales jugadas, de quién es el turno; en lugar de tener todas las variables sueltas en el programa principal, la clase Turno funciona como organizador de todos esos elementos.

■ 4.1.1. Organización del código

Una de las observaciones que se me hizo en la primera corrección fue que ningún archivo debía exceder las 400 líneas, comentario que cambió cómo organizaba mi código en gran medida. Lo que estaba “tirado” en un archivo principal tuve que repensar cómo dividirlo de forma que tuviese sentido y que funcionase sin problema. Al final me decidí por tener además del programa principal un archivo “Inicio” en el que estuviese todo lo que sucede antes de la partida y utilizar dos archivos de Python que guardasen todas aquellas funciones responsables por otras ventanas y las funciones las responsables por acciones en la partida, respectivamente. Estas segundas son en gran medida una combinación de métodos de diferentes objetos y la actualización de la interfaz gráfica bajo un nombre que resuma la acción, lo que permitió que el programa principal se leyese fácilmente.

■ 4.2. La evaluación de palabras

Uno de los requerimientos del programa es que cuando el usuario o la computadora ingresa una palabra, esta sea una palabra válida en español y que corresponda con el tipo de palabras admitido en el nivel, para lo cual se usó el diccionario de pattern.es. Un problema que surgió rápidamente fueron las palabras con tilde; el diccionario de Pattern no reconoce palabras cuando son escritas sin tilde. Las soluciones posibles eran: dejar fuera cualquier palabra con tilde, lo que hubiese limitado en gran medida la cantidad de palabras posibles; tener fichas distintas para las letras con tilde y sin tilde, aunque eso requeriría que el usuario tenga una ortografía perfecta y dificultaría en gran medida la creación de palabras; o, finalmente, encontrar una forma de que lea las palabras sin tilde correctamente.

Decidí intentar la última opción, para lo cual creé un diccionario con las palabras válidas teniendo la palabra sin tilde como clave y la palabra con tilde como valor. Para esto el programa recorre el diccionario de pattern y primero selecciona las palabras válidas (aquellas que tuviesen entre 2 y 7 letras y no tuviesen números o signos de puntuación), de la cual guarda una versión sin tilde (utilizando el módulo unicode) que asigna como clave y una versión con tilde que guarda como valor de dicha clave.

Cuando se necesita evaluar la palabra ingresada por el usuario o la computadora, entonces, solo se pasa por parámetro y se revisa que exista esa clave en el diccionario y, si existe, se revisa que el valor correspondiente a esa clave sea el tipo de palabra adecuado para el nivel (sustantivo, adjetivo o verbo) utilizando la función parse() de pattern y revisando que la etiqueta correspondiente al tipo de palabra sea la adecuada.

■ 4.3. La configuración de niveles

El programa debía tener 3 niveles de dificultad que tuviesen elementos distintos (cantidad de fichas, su puntaje, el tiempo de la partida, entre otras opciones posibles). En primera instancia realicé un diccionario nivel “fácil”, uno “medio” y uno “difícil” que tuviese todas las variables que dependieran de la dificultad, que luego guardaba en tres archivos json y uno de los cuales se abría en el programa principal según lo elegido por el usuario. Este método probó ineficiente cuando quise realizar un tipo de nivel “customizado”, es decir, que el usuario pudiese decidir de cada característica posible el nivel de dificultad. En la segunda formulación, entonces, armé un diccionario “nivel” que contenía para cada característica su propio diccionario anidado con las variaciones de las tres dificultades.

En el inicio, entonces, siempre se carga un archivo con un único diccionario y se crea un diccionario nuevo llamado “config” el que guarda todas las especificaciones del nivel actual de forma que sea sencillo acceder a ellas. Si el usuario no elige ningún nivel, cargará por defecto el nivel fácil y, si elige la customización, cualquier categoría que deje sin elegir quedará en fácil.

Otro problema en el proceso fue la necesidad de poder mostrarle al usuario de manera sencilla su configuración actual; para lo cual la solución fue crear una lista que tuviese en forma de texto cada configuración. Algunas variables como “Dificultad Computadora” o “Puntaje de las fichas” se denominan solamente con la palabra correspondiente al nivel, pero otras como “Tiempo” y “Tipo de palabras permitidas” se reinterpretan a textos como “2 minutos” y “sustantivos, adjetivos y verbos”.

■ 4.3.1. La ventana de configuración

Para configurar un nivel customizado en el principio o para cambiar la configuración durante la partida, se abre una ventana nueva de configuración (ver figura 3 en la página 16), la cual muestra opciones para cambiar de nivel o configurar manualmente uno mediante una serie de Listbox. Si se cambia la configuración, se actualiza en el diccionario “config” las variables necesarias. En el caso de la bolsa con fichas, primero debe vaciarse de las que ya estaban para luego agregar las nuevas fichas; para el tiempo de la partida se resta al nuevo tiempo el ya transcurrido; y si se cambia el tablero se actualiza de forma que los colores reflejen los nuevos valores de cada ficha.

■ 4.4. Pausa y reanudación de la partida

Uno de los botones en la interfaz principal llamado “Guardar Partida” permite al usuario guardar la partida actual en su estado, para después reanudarla con el botón “Continuar” en la ventana de Inicio.

Los primeros elementos a guardar fueron simples: el tablero con los casille-

ros ocupados, nombre, puntaje y atril de los jugadores, la bolsa de fichas y la configuración de nivel. Para esto, los objetos Tablero, Jugador y Computadora (por extensión de Jugador) tienen métodos que guardan y devuelven la información actual, la cual se guarda en un diccionario “juego” y se guarda en un archivo json. A la hora de reanudar la partida se abre ese archivo, si es que existe, y se actualizan los datos a aquellos recuperados. Sin embargo, en el proceso se hizo evidente que faltaba guardar algunos elementos clave.

Primero, las fichas ubicadas anteriormente se guardaban y cargaban correctamente, pero no se pintaban los casilleros del color correspondiente a quién había ubicado cada una. Para esto se creó una variable de instancia en el objeto Jugador llamada “_casilleros_usados” la cual guarda cada tupla correspondiente a cada casillero utilizado por el jugador. Guardando esta variable en el diccionario de juego, cuando se reanuda la partida se puede acceder a este dato para pintar cada casillero del color correspondiente.

Otro error que apareció fue en el caso de que el jugador pausase la partida con fichas ubicadas antes de terminar el turno, estas quedaban cargadas en la matriz del tablero. Para esto solo fue necesario agregar cuando se ejecutase la opción de guardar partida el método dentro de Turno llamado “limpiar”, que se encarga justamente de sacar todas las letras no bloqueadas del tablero y desbloquearlas en el atril.

Seguidamente, fue necesario agregar al diccionario cuando se implementaron los popups de palabras jugadas y configuración actual, las variables que contuviera esa información y, por último, pasar la variable que guarde si es el turno de la computadora o del jugador, en orden de no saltar el turno de ninguno.

■ 4.5. Tableros generativos

La disposición de los tipos de fichas en el tablero debía ser de la misma forma que el diseño de un tablero de Scrabble tradicional: un patrón que se espeje en ambos sentidos. Para los 3 niveles se realizó recorriendo una matriz de 15x15 y mediante condiciones anidadas especificar el tipo de ficha correspondiente, ya que eso permitía más control sobre la cantidad de fichas de cada tipo por nivel. Sin embargo, estos tres tableros son invariables, por lo que me pregunté si podría haber una forma de crear tableros generativamente, creando una mayor variabilidad.

La solución a la que llegué no me permite tanto control sobre la cantidad de tipo de fichas y produjo tableros en los que la mayoría de las fichas eran especiales, creando algunos tableros llenos de puntos extra o extremadamente difíciles. Quedó entonces no como el generador de tableros para los niveles, sino como un creador aparte, solo disponible cuando desde la customización se elige la dificultad de tablero “Sorpresa”.

Esta idea se basa en el Juego de la vida de John Conway⁴: un objeto Celda que empieza con un estado de 0 o 1 y que en cada nueva generación actualiza su valor según el estado de vida de sus adyacentes. Las celdas forman una grilla de 15x15 doblemente espejada y hay un objeto Grilla que guarda una versión del tablero en todas sus épocas y actualiza una nueva generación hasta que la suma de cada tablero (es decir, el tablero que queda de la suma la Celda[x,y] de todos los juegos hasta quedar con un solo tablero) devuelva por lo menos una Celda de valor 8. Esto es porque hay 8 tipos de casilleros distintos, por lo que nos aseguramos de esta forma que haya por lo menos una ficha de cada tipo. Cuando terminan las iteraciones se asigna a cada número de un tipo de casillero y queda formado el tablero.

5. Consideraciones éticas sobre el desarrollo

5.1. Accesibilidad del programa

Para poder ejecutar este programa se necesita por lo menos un conocimiento básico de Python y una computadora con Windows o Linux, ya que se debe tener instalado el programa y las librerías Pattern y PySimpleGUI con la versión correcta. El funcionamiento en Apple no está probado, por lo que no puede garantizarse su correcto funcionamiento en esa plataforma.

En cuanto al programa en sí, requiere principalmente que el usuario no tenga dificultades de motricidad fina, dado la cantidad de botones pequeños situados en cercanía uno de otros. Por otra parte, aunque la mayoría de los colores utilizados en el diseño de la interfaz contrastan, los dos colores utilizados para diferenciar las palabras utilizadas por el usuario y la computadora pueden causar problemas para personas con cierto tipo de daltonismo sensible a los tonos azules.

5.2. Privacidad de la información

ScrabbleAR no guarda ningún tipo de información sobre el usuario además de la partida que se pueda guardar o los datos que se solicitan para el Top10, de los cuales solo toma de la computadora la fecha.

5.3. Licencias de los software de desarrollo

Para el desarrollo de esta aplicación se utilizó Python, un software open-source, junto a Pattern, con licencia BSD; PySimpleGUI, con Licencia Pública General Reducida de GNU (LGPL de GNU) y PyCharm, cuya versión gratuita es open-source.

5.4. Licencia del software desarrollado

El código desarrollado para este proyecto, junto con las piezas gráficas están

⁴ Wikipedia. (n.d.). Juego de la vida. Wikipedia. https://es.wikipedia.org/wiki/Juego_de_la_vida

disponibles para ser vistas, descargadas y modificadas en el siguiente repositorio de GitHub <https://github.com/juliasaenz/ScrabbleAR>

■ 5.5. Documentación generada

Además de tener comentarios cómo documentación en el código, este proyecto cuenta con una Guía de Usuario y una Guía para el desarrollador disponibles para descargar junto al proyecto en GitHub.

■ 6. Conclusiones y trabajos futuros

■ Esta materia y este trabajo final se desarrolló en su enteridad en el contexto del Distanciamiento Social, Preventivo y Obligatorio necesario por la pandemia del virus Covid-19 y, en consecuencia, la cursada se realizó casi exclusivamente en modalidad virtual. Para el trabajo final esto causó ciertas complicaciones: poco después de empezar el trabajo final, mis dos compañeros de trabajo dejaron la materia por diversas complicaciones, por lo que terminé realizando una consigna normalmente dada para 2 o 3 alumnos yo sola. Esto fue a su vez una desventaja, ya que tenía que hacer por lo menos el doble del trabajo, además de que no me dio la oportunidad de experimentar con tener un grupo de trabajo con el cual debía comunicarme y organizarme; pero también fue una bendición desde el punto que tuve que aprender a utilizar todas las herramientas que quisiese usar y también pude trabajar en mis propios tiempos, sin necesidad de reportar a nadie.

El no tener que reportar a nadie no es estrictamente cierto, ya que tenía un ayudante al cual podía consultar todas las semanas en las clases, o incluso por mensaje si lo necesitase. Reflexionando sobre el proceso, puedo decir que no aproveche esta oportunidad de corrección, participando poco en las clases que no fuesen de entrega obligatoria. Esto me jugó en contra y fue evidente en la corrección final que, aunque funcionaba correctamente, tenía algunas cuestiones (como poder ingresar número negativo como puntaje de las fichas) que no estaban solucionadas por el simple hecho de que no se me había ocurrido buscar por esas fallas.

Una de las mejores decisiones que tomé en el proceso fue una de las primeras: la de trabajar con objetos y organizar desde el principio qué debía hacer cada uno. Esto me permitió tener siempre una base sobre la cual trabajar, que pudiese expandirse y modificarse fácilmente, intentando también que fuese legible no solo para mi sino para cualquier otro programador que leyese mi código y que fuese eficiente. Junto a esto, la decisión de ir documentando el código a medida que lo escribía me ahorró en muchos casos tener que releer código preguntándome qué hacía una sección en particular.

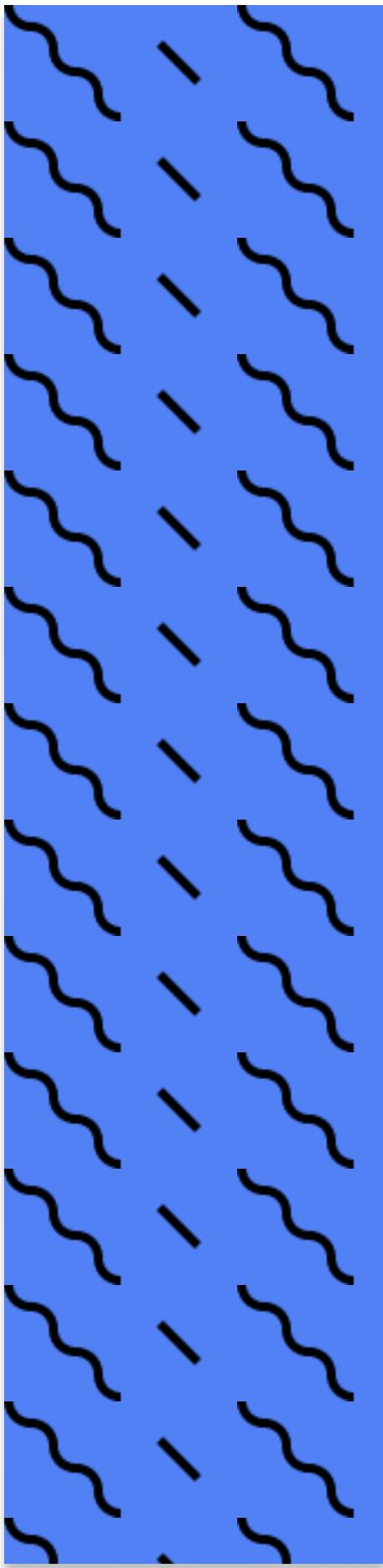
Nunca había programado en Python previo a esta materia; aprender este

lenguaje en el transcurso de la materia y en el trabajo final no solo me dió las herramientas básicas para seguir adentrándome en este lenguaje, sino que me permitió ver la inmensa potencialidad del mismo. Como todo programador, tuve que ir en ocasiones a buscar en internet ayuda o soluciones, y en cada caso no solo encontré una comunidad enorme con tutoriales y recursos, sino que aparecían proyectos, objetos y aplicaciones realizados con Python usando librerías o paquetes de todo tipo dando cuenta de la infinidad de cosas que realizarse con este lenguaje.

En cuanto a trabajos futuros, tengo intención de seguir trabajando con Python en cada proyecto que tenga la oportunidad. Actualmente, por ejemplo, estoy aprendiendo para un trabajo de clase, la librería matplotlib para visualizar datos. Una de las cuestiones que más me interesan, ya que además de la carrera de Analista en Programador Universitario estoy terminando la carrera de Diseño Multimedial también en la UNLP, es el diseño de interfaces, particularmente interfaces gestuales, para lo cual estoy investigando el uso de la librería OpenCV para la detección de gestos de la mano; y pienso también en indagar sobre el uso y entrenamiento de redes neuronales con la esperanza de realizar un proyecto utilizando estas tecnologías que, de llevarlo a cabo, estará disponible en mi GitHub.

7. Referencias

- Agasthyan, R.(2017) envatotuts+: “Entendiendo Entornos Virtuales en Python”. Traducción por Rafael Chavarría. Recuperado de:
<https://code.tutsplus.com/es/tutorials/understanding-virtual-environment-s-in-python--cms-28272>
- Documentación de Conda. Disponible en: <https://docs.conda.io/en/latest/>
- Documentación de Pattern. Disponible en: <https://github.com/clips/pattern/wiki>
- Documentación de pip. Disponible en: <https://pip.pypa.io/en/stable/>
- Documentación de PySimpleGUI. Disponible en:
<https://pysimplegui.readthedocs.io/en/latest/cookbook/>
- Documentación de Python sobre entornos virtuales y paquetes. Disponible en:
<https://docs.python.org/3/tutorial/venv.html>
- Documentación de virtualenv. Disponible en: <https://virtualenv.pypa.io/en/stable/>
- Drisoll, M. (2020). Real Python: “PySimpleGUI: The Simple Way to Create a GUI With Python”. Recuperado de: <https://realpython.com/pysimplegui-python/>
- García Martínez Montserrat. (2015). Concepto de ética informática. Ensayo. Recuperado de
<https://www.gestiopolis.com/concepto-de-etica-informatica/>
- PySimpleGUI. Descripción del proyecto. Disponible en:
<https://pypi.org/project/PySimpleGUI/>
- Repositorio de Pattern. Disponible en <https://github.com/clips/pattern>
- Repositorio de PySimpleGUI. Disponible en:
<https://github.com/PySimpleGUI/PySimpleGUI>
- Sitio oficial de PyCharm. Disponible en: <https://www.jetbrains.com/pycharm/>
- Wikipedia. (n.d.). Juego de la vida. Wikipedia.
https://es.wikipedia.org/wiki/Juego_de_la_vida



Anexo A: Guía de Usuario

■ Antes de jugar

Para poder ejecutar ScrabbleAR se debe tener

- Windows o Linux
- Python versión 3.6
- PySimpleGUI versión 4.19.0
- Pattern versión 3.6

■ Inicio

Se ejecuta el archivo llamado “ScrabbleAR” y aparecerá la siguiente ventana:

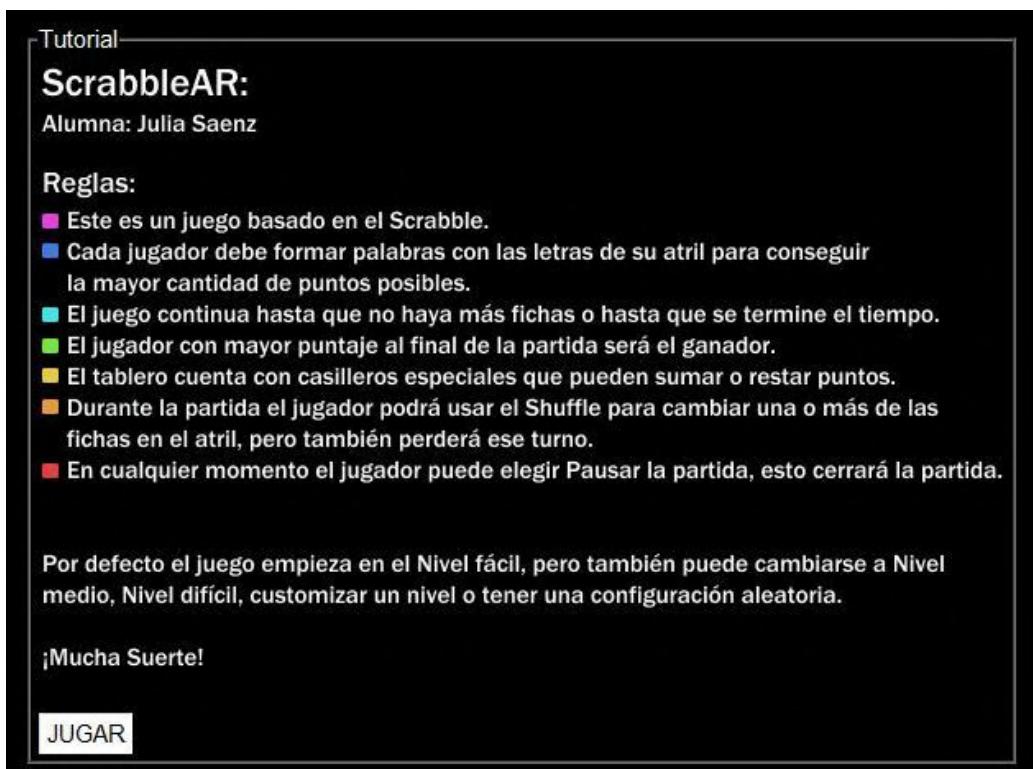


Figura 1: Ventana de tutorial

Luego de leer las reglas se presiona el botón “JUGAR”, que lo llevará a la siguiente ventana:



Figura 2: Ventana de Inicio

En el espacio llamado “TU NOMBRE” debes ingresar el nombre con el que quieras jugar la partida. Este debe tener por lo menos un carácter y máximo 15 caracteres. Luego se puede elegir la dificultad del nivel. “FÁCIL”, “MEDIO” y “DIFÍCIL” son las opciones predeterminadas pero también se puede elegir “ALEATORIO”, que elegirá dificultad para cada variable al azar, o “CUSTOMIZADO” que le permitirá configurar el nivel de dificultad para cada elemento. En caso de no completar el nombre, su nombre de usuario será “TU NOMBRE” y si no se elige un nivel empezará por defecto en el nivel “FÁCIL”.

NUEVA PARTIDA comenzará un nuevo juego, mientras que el botón CONTINUAR solo estará disponible si hay una partida guardada con anterioridad. Si se elige CONTINUAR se abrirá la última partida guardada, con su respectivo nombre de jugador y configuración de dificultad.

Si se selecciona el nivel “CUSTOMIZADO” se abre una nueva ventana en la que se puede seleccionar la dificultad de cada elemento, el nivel general y, si quiere, la cantidad o puntaje de cada letra. Para guardar la configuración y empezar la partida se presiona “CONFIRMAR CAMBIOS”

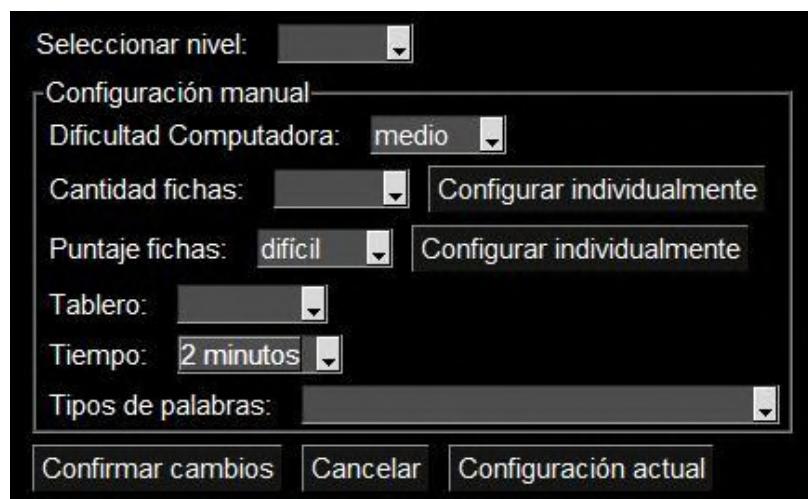


Figura 3: Ventana de Configuración

A	E	I	O	U	B	N	D	F	L	M	P	G	H	V	C	R	S	J	K	Ñ	Q	T	W	X	Y	Z
12	12	6	9	5	2	5	5	1	4	2	2	2	2	1	4	5	6	1	1	1	1	4	1	1	1	1

Lista **Cancelar**

Figura 4: Configuración de letras individual

■ La partida

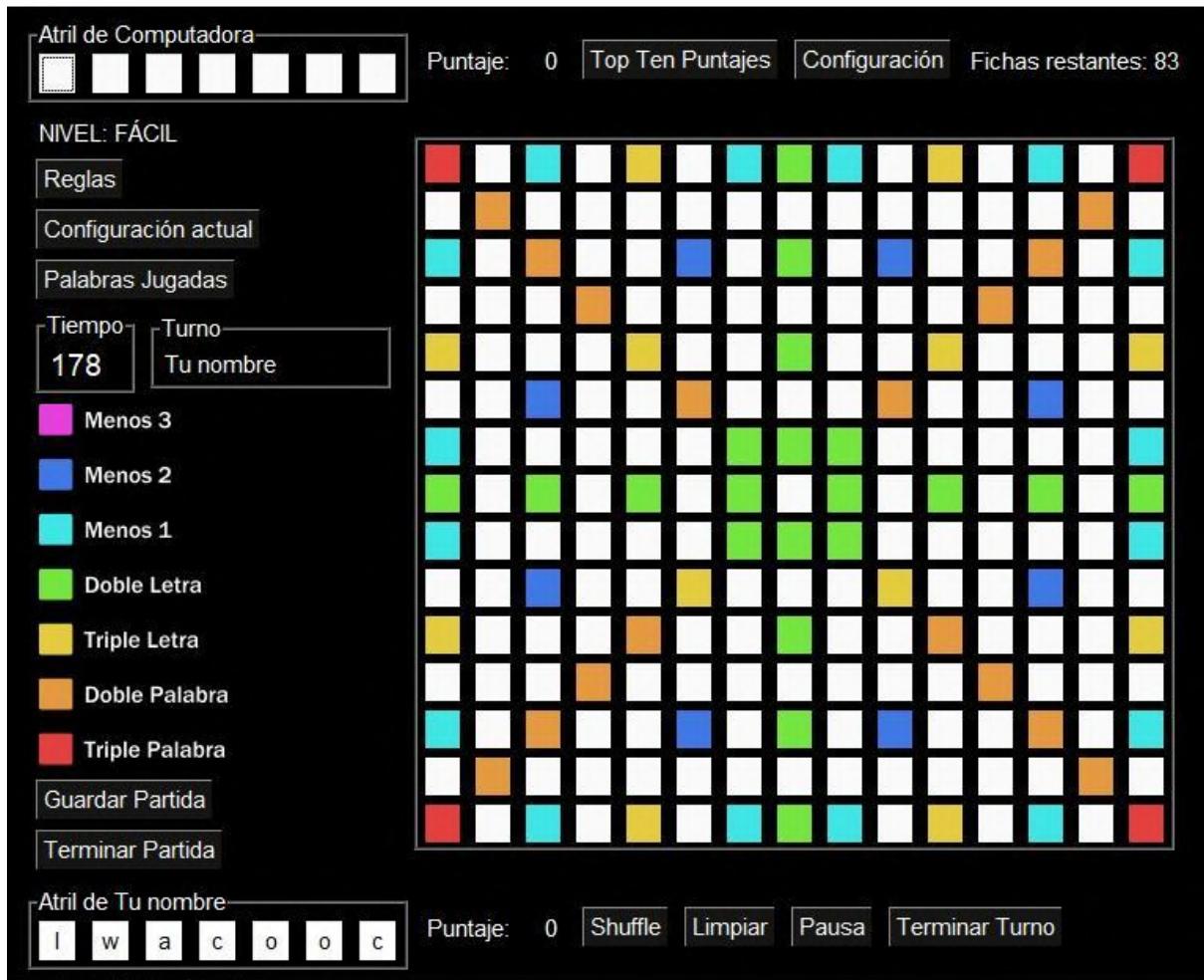


Figura 5: Ventana Principal

La figura 4 muestra una captura de la partida. De la esquina superior izquierda a la esquina inferior derecha se puede observar.

- El atril de la computadora: No se pueden ver las fichas que tiene el atril
- Puntaje de la Computadora: muestra los puntos actuales de la computadora
- Top Ten Puntajes: Muestra un PopUp con las diez mejores partidas jugadas. Si el historial tiene menos de 10 partidas muestra las que haya guardadas.

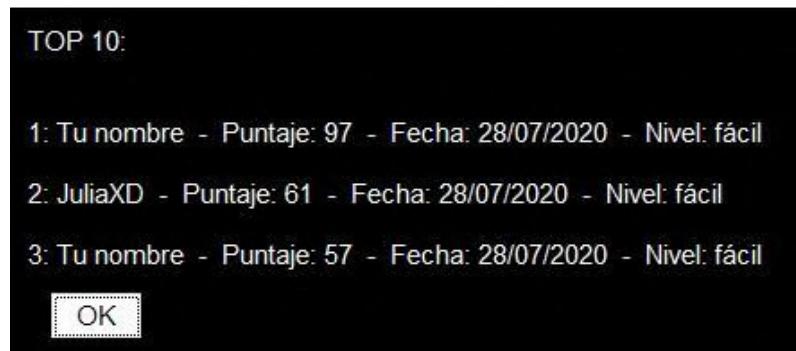


Figura 6: PopUp de Top 10

- Configuración: Permite configurar la dificultad de la partida, si acaso lo desea
- Fichas restantes: Indica cuántas fichas quedan en la bolsa. Si se terminan las fichas se termina la partida
- Nivel: muestra la dificultad en la que se está jugando
- Reglas: Muestra un PopUp con las reglas de la partida
- Configuración actual: Muestra un PopUp especificando la configuración de dificultad actual de la partida

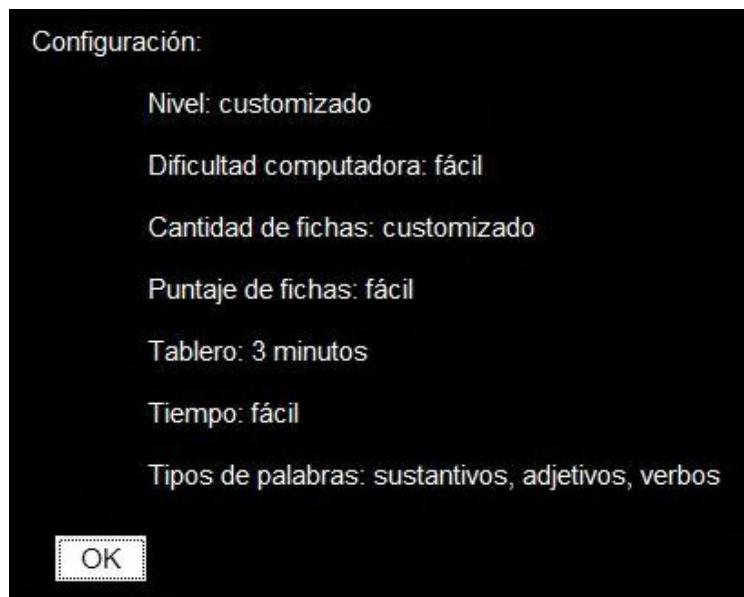


Figura 7: PopUp de Configuración Actual

- Palabras Jugadas: Muestra un Popup con la lista de palabras jugadas por jugador en orden y el puntaje de cada una

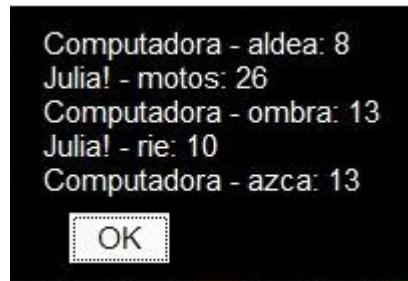


Figura 8: PopUp de Palabras Jugadas

- Tiempo: Un contador del tiempo restante de la partida en segundos, cuando llega a 0 termina la partida
- Turno: Texto que especifica quién está jugando
- Guía: Guía de los puntajes correspondientes a cada color de casileto
- Guardar Partida: Si se guarda la partida esta se cierra y se guarda de forma que cuando se vuelve al Inicio se puede seleccionar CONTINUAR para seguirla. Solo se guarda de a una partida a la vez, por lo que si ya tiene una partida guardada y vuelve a seleccionar GUARDAR PARTIDA se sobreescrivira
- Terminar Partida: Si se confirma, termina la partida y anuncia el ganador
- Atril de Jugador: Muestra las fichas del atril del jugador
- Puntaje Jugador: Muestra los puntos acumulados del jugador
- Shuffle: Permite seleccionar entre 1 y 7 fichas de tu atril para intercambiarlas por fichas aleatorias de la bolsa. Esto puede hacerse 3 veces por partida y hacerlo significa saltar un turno.

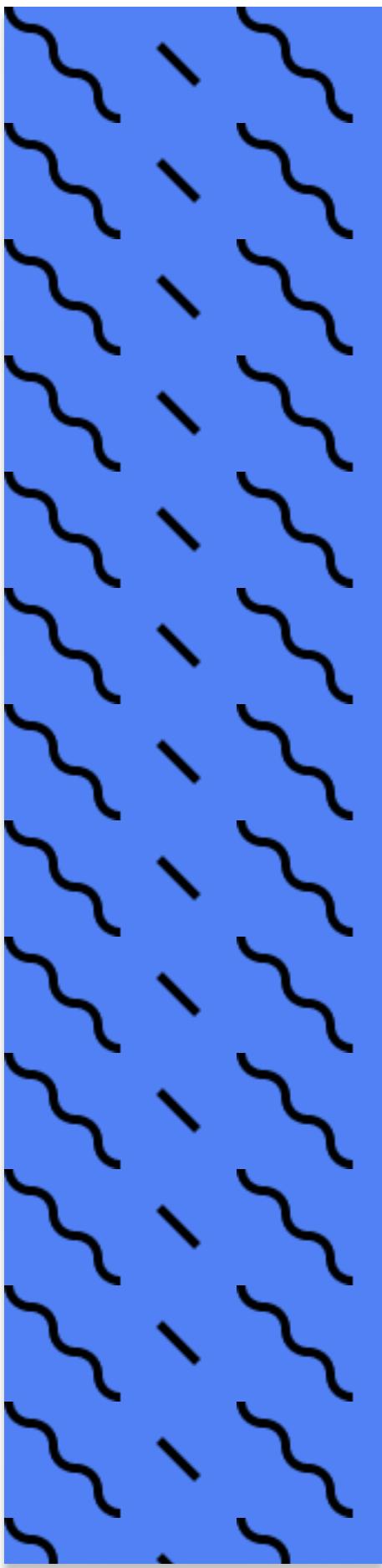


Figura 9: PopUp de Shuffle

- Limpiar: Devuelve todas las fichas ubicadas en el turno actual (no las que se utilizaron en turnos anteriores) al atril
- Pausa: Pausa el juego
- Terminar Turno: Revisa que la palabra sea válida y, si lo es, pasa a ser el turno de la computadora. Si la palabra no es válida vuelven todas las fichas ubicadas en el turno al atril y sigue siendo el turno del jugador

El juego puede terminar de tres formas: Que se acabe el tiempo de la partida, que no haya suficientes fichas en la bolsa para que ambos jugadores tengan 7 fichas o que el jugador presione el botón TERMINAR PARTIDA. En todos estos casos el programa mostrará quién es el ganador sumando el puntaje de todas las palabras jugadas y restando a cada uno el valor de las fichas que les quedaron en el atril. Se mostrará el resultado final en un PopUP y, si el Jugador resultó ganador y su puntaje es lo suficientemente alto para entrar en el Top10 de partidas, se le dará la opción de guardar su partida en el Top10.

Luego de terminada una partida volverá a aparecer la ventana de Inicio y podrá jugar una nueva partida.



Anexo B: Guía para el desarrollador

ScrabbleAR

1.0

Julia Saenz

04 de octubre de 2020

CAPÍTULO 1

Inicio del Programa

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

`Inicio.correr_inicio()`

Abre la ventana de inicio y, de ser necesario, inicializa las variables de la partida

`Inicio.correr_tutorial()`

Abre la ventana de Tutorial

CAPÍTULO 2

Clases

2.1 Casillero

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

```
class Clases.Casillero.Casillero(tipo_='normal')  
    _tipo El tipo de casillero que es  
    _letra La letra que ocupa ese casillero  
    _bloqueado Los casilleros usados al fin del turno se bloquean y no pueden modificarse  
  
    bloquear()  
        Pone self._bloqueado en True  
  
    devolver_puntos(puntos, palabra_)  
        Depende del tipo de casillero devuelve los puntos correspondientes  
  
    dibujar(clave)  
        Recibe la clave del botón y devuelve el botón según el tipo de casillero  
  
    doble_letra(puntos)  
        Devuelve el doble puntaje de la letra  
  
    doble_palabra(puntos, pal)  
        Devuelve el doble puntaje de la palabra  
  
    esta_bloqueado()  
        Devuelve si el Casillero está bloqueado  
  
    get_letra()  
        Devuelve la letra en el Casillero  
  
    get_tipo()  
        Devuelve el tipo de Casillero
```

menos_dos (*puntos*)
Resta 2 a la letra

menos_tres (*puntos*)
Resta 3 a la letra

menos_uno (*puntos*)
Resta 1 a la letra

set_letra (*letra*)
Actualiza la letra en el Casillero

set_tipo (*tip*)
Actualiza el tipo de Casillero

triple_letra (*puntos*)
Devuelve el triple puntaje de la letra

triple_palabra (*puntos, pal*)
Devuelve el triple puntaje de la palabra

2.2 Tablero

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

```
class Clases.Tablero.Tablero (tipos)
```

Todos los cambios que se quieran a hacer a un Casillero deben hacerse a través de los métodos en Tablero

_dimension Cantidad de casilleros de alto y ancho que tiene el tablero

_matriz Arreglo de Casilleros de tamaño **_dimension** x **_dimension**

_posiciones Arreglo de tuplas de posición de tamaño **_dimension** x **_dimension**

Tipos Arreglo de strings que especifican el tipo de casillero de tamaño **_dimension** x **_dimension**

actualizar_casillero (*letra, pos*)
Recibe una letra y una posición y actualiza ese Casillero

actualizar_tipo (*pos, tip*)
Recibe una letra y una posición y actualiza el tipo de ese Casillero

armar_posiciones ()
Crea una matriz de tuplas con las posiciones para los botones

bloquear_casilleros (*casilleros*)
Recibe una letra y una posición y bloquea ese Casillero

continuar_partida (*arreglo*)
Recibe una matriz con los casilleros ocupados y actualiza el Tablero

dibujar ()
Devuelve un arreglo de **_dimension** x **_dimension** de botones

esta_bloqueado (*pos*)
Recibe una letra y una posición y devuelve si ese Casillero está bloauqeado

```

get_casillero (pos)
    Recibe una letra y una posición y devuelve ese Casillero

get_matriz ()
    Devuelve el arreglo de la matriz

get_posiciones ()
    Devuelve la matriz de tuplas de posiciones

limpiar_matriz ()
    Elimina de la matriz cualquier ficha que no esté bloqueada

pausar_partida ()
    Guarda los valores de la matriz y los devuelve

```

2.3 Jugador

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

```

class Clases.Jugador.Jugador (nom)

    Bolsa Variable de Clase. Arreglo de letras en el juego

    _nombre El nombre del jugador

    _puntaje Puntos acumulados en la partida

    _atril Arreglo de letras en el atril del jugador

    Cambios Cantidad de cambios posibles del usuario

    _casilleros_usados Posiciones de los casilleros con letras puestas por el usuario

    actualizar_puntaje (pun)
        Suma al puntaje total los puntos nuevos acumulados

    add_casilleros_usados (cas)
        Agrega a un arreglo la posición de un casillero

    armar_atril ()
        Crea el arreglo de 7 fichas aleatorias sacadas de la bolsa

    continuar_turno (datos)
        Recibe y actualiza el puntaje, cambios y atril del jugador

    dibujar ()
        Devuelve un arreglo de botones con las fichas del atril

    fin_de_turno (puntos, usadas, cas)
        Actualiza el puntaje, agrega los casilleros usados y repone el atril

    get_atril ()
        Devuelve las fichas del atril

    get_cambios ()
        Devuelve la cantidad de cambios hechos en la partida

    get_cant_bolsa ()
        Devuelve la cantidad de fichas en la bolsa

```

get_ficha (pos)
Devuelve la ficha correspondiente a una posición del atril

get_nombre ()
Devuelve el nombre del jugador

get_posicion_letra (letra)
Devuelve la posición en el atril de una letra

get_puntaje ()
Devuelve el puntaje del jugador

guardar_partida (nivel)
Guarda y devuelve el nombre y puntaje del jugador, la fecha actual y el nivel al que jugó

pausar_turno ()
Guarda y devuelve el nombre, puntaje, atril, cambios y casilleros usados del jugador

reponer_atril (usadas)
Repone el atril según la cantidad de fichas que se hayan usado en el turno

sacar_fichas (fichas)
Saca una ficha del atril

set_puntaje (pun)
Setea el puntaje del jugador

shuffle (fichas)
Elimina las letras seleccionadas del atril y repone con fichas aleatorias de la bolsa

terminar_partida (puntos)
Resta del puntaje total el valor de las fichas en el atril

2.4 Computadora

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

class Clases.Computadora.Computadora
Extiende Jugador, por lo que tiene todos sus variables y métodos

_palabra Guarda la palabra elegida por la computadora cada turno

_casilleros Arreglo de posiciones de los casilleros usados en el turno

_atril_usadas Arreglo de las letras usadas del atril

_puntos Puntos acumulados en un turno

_max Longitud de la palabra mas larga posible en un turno

definir_puntos (matriz, puntos, casilleros)
Cuenta la cantidad de puntos que suma una palabra

dibujar ()
Dibuja el atril de la compu como botones desactivados

get_casilleros ()
Devuelve los casilleros en que utiliza la palabra

```

get_palabra()
    Devuelve la palabra creada por la Computadora

get_puntaje_palabra()
    Devuelve el puntaje de la palabra

jugada(matriz, diccionario, nivel, primer_turno)
    Arma la palabra y la ubica en el Tablero

sacar_y_reponer_atril()
    Saca las fichas del atril, repone fichas y borra el valor de la palabra y puntos

```

2.5 Turno

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

```

class Clases.Turno.Turno

    _letras Letras usadas en el turno (sin orden)
    _palabra La palabra que se forma
    _casilleros_usados Guarda las posiciones de los casilleros usados
    _letra_actual Guarda el valor de la letra actual
    _pos_actual El último valor del atril presionado
    _orientacion La orientación de la palabra
    _turno_usuario_ True si es el turno del usuario
    _lista_palabras Lista de palabras jugadas en el la partida
    _puntaje Los puntos de la palabra jugada en el punto
    Primer_turno True hasta que se juegue el primer turno

add_atril_usada(num)
    Agrega a un arreglo las letras del atril usadas

add_lista_palabras(palabra, puntos, nom)
    Agrega una palabra, su puntaje y quién la jugó a la lista de palabras jugadas

agregar_casillero(pos)
    Agrega los casilleros usados en el turno, si hay más de dos guardados, calcula la orientación
    de la palabra

definir_palabra(matriz)
    Según la orientación, ordena los casilleros usados, recupera las letras correspondientes y
    evalúa si la palabra es válida

definir_puntos(matriz, puntos)
    Toma los casilleros usados para sumar el puntaje total de la palabra

es_turno_usuario()
    Devuelve si es el turno del usuario

evaluar_palabra(matriz, diccionario, nivel)
    Se fija si la palabra es válida y en ese caso devuelve sus puntos

```

get_atril_usadas()
Devuelve las letras del atril usadas

get_casilleros_usados()
Devuelve los casilleros usados

get_letra_actual()
Devuelve la última letra del atril seleccionada

get_letras()
Devuelve las letras usadas

get_lista_palabras()
Devuelve la lista de palabras usadas como texto

get_orientacion()
Devuelve la orientación de la palabra

get_palabra()
Devuelve la palabra

get_pos_actual()
Devuelve el último casillero del atril seleccionado

get_primer_turno()
Devuelve si es el primer turno de la partida

guardar_lista_palabras()
Devuelve la lista de palabras

jugue_primer_turno()
Marca el primer turno como jugado

limpiar()
Reinicia todas las variables del turno

reinicio(nom)
Agrega la palabra jugada a la lista, reinicia los valores de turno y actualiza de quién es el turno

sacar_casillero(pos)
Saca el casillero indicado de los casilleros usados

set_casilleros_usados(c)
Actualiza los casilleros usados a una lista pasada por parámetro

set_letra_actual(letra)
Actualiza la última letra del atril seleccionada

set_letras(letra)
Actualiza las letras usadas

set_lista_palabras(lista)
Actualiza la lista de palabras con una lista pasada por parámetro

set_palabra(p)
Actualiza la palabra a una pasada por parámetro

set_pos_actual(pos)
Actualiza el último casillero del atril seleccionado

set_puntaje (p)
Actualiza el puntaje con un valor pasado por parámetro

set_turno_usuario (va)
Actualiza si es el turno del usuario

validar_turno ()
Se fija, si es el primer turno, que haya una ficha en el casillero del medio

2.6 Tableros Generativos

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

```
class Funciones.tableros_extra.Celda (estado=1)

Estado Puede ser 1 o 0
_estado_futuro Guarda cual será el próximo estado de la Celda

actualizar_celda (celdas, i, j)
    Actualiza el estado_futuro de la celda según el estado de sus vecinas

recargar ()
    Pasa su estado_futuro a su estado actual

class Funciones.tableros_extra.Grilla (cantidad_juegos_: object, cantidad_epocas_: object)

Juegos Lista de Juegos de La Vida
Cuantos_necesito Cual es el máximo número al que necesito llegar en por lo menos una celda
Cantidad_juegos Cantidad de Juegos de La Vida que quiero
Cantidad_epocas Cantidad de épocas para cada Juego de La Vida
_continuar Si es True, sigue creando Juegos

grilla_final ()
    Traduce toda la grilla final de números a strings de tipo de Casillero y devuelve esa matriz

hay_suficientes ()
    Se fija si hay por lo menos una celda que, sumando todos los juegos, valga 8

imprimir ()
    Imprime la grilla final

class Funciones.tableros_extra.JuegoDeLaVida
    Objeto basado en el Juego de la Vida de John Conway para armar tableros generativos

actualizar ()
    Actualiza todas las celdas de la matriz y las recarga

armar_celdas ()
    Inicia en cada espacio de la matriz una Celda que vale 0 o 1 aleatoriamente de forma que el patrón sea doblemente simétrico

hay_vivas ()
    Devuelve si hay en la matriz más de 20 celdas vivas (que valgan 1)
```

imprimir()

Imprime el estado de las celdas

Funciones.tableros_extra.**tablero_aleatorio()**

Genera un tablero aleatorio

CAPÍTULO 3

Funciones

3.1 Validación de Palabras

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

`Funciones.funciones_palabras.existe_palabra(palabra, diccionario)`

Devuelve True si la palabra sin tilde existe entre las claves del diccionario

`Funciones.funciones_palabras.palabra_es_valida(palabra, diccionario, lista)`

Se ingresa la palabra y la cantidad de categorias para chequear

nivel fácil = ingresar 1, 2

nivel medio = ingresar 2

nivel difícil = ingresar 2

`Funciones.funciones_palabras.palabras_sin_tilde()`

Crea un diccionario que tiene de clave las palabras sin tilde y de valor la palabra con tilde(en caso de tenerla). Solo agrega palabras entre 2 y 7 letras que no tengan números o signos de puntuación

`Funciones.funciones_palabras.tiene_estas(palabra)`

Devuelve si la palabra pasada por parámetro incluye un número o signo de puntuación

3.2 Funciones de Partida

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

`Funciones.funciones_partida.letra_actual(event, turno, jugador, window)`

La letra del atril seleccionada por el usuario

`Funciones.funciones_partida.limpiar(turno, tabla, window)`

Saca las fichas no bloqueadas del tablero y las vuelve a activar en el atril

Funciones.funciones_partida.**pausar** (*turno, jugador, compu, tabla, window, config, bolsa, act_config*)

Guarda los datos de los jugadores, el tablero, la bolsa y el nivel en un archivo

Funciones.funciones_partida.**poner_ficha** (*event, turno, tabla, window*)

Ubica una ficha en el tablero

Funciones.funciones_partida.**que_color** (*tipo*)

Recibe un String con el tipo de casillero y devuelve el color correspondiente

Funciones.funciones_partida.**reinicio_partida** (*window, config, tiempo, Jugador, turno, jugador, compu, diccionario, act_config, continuar, tabla, niveles*)

Elimina las variables usadas para reiniciar la partida

Funciones.funciones_partida.**shuffle** (*turno, tabla, jugador, window*)

Cambia las fichas del atril seleccionadas por el usuario y saltea el turno, se puede hacer 3 veces por partida

Funciones.funciones_partida.**terminar_partida** (*jugador, compu, window, config, nivel*)

Se cuentan los puntos finales y se muestra el ganador, si el usuario gana, le da la opción de guardar el puntaje

Funciones.funciones_partida.**terminar_turno** (*turno, tabla, jugador, window, diccionario, config*)

Si la palabra es correcta pasa al turno de la compy, sino limpia el tablero

Funciones.funciones_partida.**top_10** ()

Abre un archivo JSON con los 10 mejores puntajes guardados, ordenados de mayor a menor y lo muestra en forma de PopUp

Funciones.funciones_partida.**turno_compu** (*turno, tabla, compu, window, config, diccionario*)

La computadora elige palabra y la posiciona según el nivel

3.3 Ventanas Secundarias

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

Funciones.Ventanas_secundarias.**actualizar_todo_dicc** (*config, niveles, dificultad, tiempo, bolsa, act_config*)

Actualiza los datos de la configuración actual según la nueva dificultad seleccionada

Funciones.Ventanas_secundarias.**configurar_dificultad** (*config, niveles, bolsa, tiempo, act_config*)

La configuración de la dificultad: por niveles o personalizada

Funciones.Ventanas_secundarias.**configurar_letras** (*dicc*)

Permite cambiar por letra su cantidad en la bolsa o su puntaje

Funciones.Ventanas_secundarias.**inicio** ()

Ventana de inicio

Funciones.Ventanas_secundarias.**tutorial**()

Ventana de tutorial

Funciones.Ventanas_secundarias.**ventana_shuffle**(atril)

Permite elegir qué fichas del atril intercambiar

3.4 Tablero Top10

Funciones.leaderboard.**entra_en_top**(puntaje)

Calcula si el puntaje de la partida es mayor al último de los mejores 10 puntajes

Funciones.leaderboard.**guardar_partida**(jugador)

Guarda el nombre y puntaje del usuario junto con la fecha y el nivel en el Top10

CAPÍTULO 4

Otros

4.1 Creación de Niveles

Trabajo para Seminario de Python 2020 - Alumna Saenz Julia

Funciones.Niveles.**categorias** = {'difícil': [2], 'fácil': [1, 2], 'medio': [2]}
Diccionario de elementos que configuran un nivel

4.2 Estilo de Interfaz

Variables de estilos de los botones, textos y títulos de la interfaz