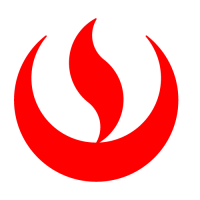
**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**FACULTAD DE INGENIERÍA**

**CARRERA PROFESIONAL DE CIENCIAS DE LA COMPUTACIÓN**



**Asignatura:**

**CC184- COMPLEJIDAD ALGORÍTMICA**

**Sección: CC41**

**Informe de trabajo final:**

**JUEGO UBONGO**

**Autor**

Montes Zarria, Martín André

**Profesor**

Teofilo Chambilla Aquino

*Campus Monterrico,*

*Julio de 2020*

**INFORME PRÁCTICO**

# DATOS GENERALES

1. Estudiante : Martín André Montes Zarria Calificación: \_\_\_\_\_\_\_
2. Práctica : Semana 15
3. Asignatura : Complejidad Algorítmica Sección: CC41
4. Profesor : Teofilo Chambilla Aquino
5. Archivo : U20181F751\_Montes.rar

# INTRODUCCIÓN

Elabore la introducción contestando estas preguntas

* + Este proyecto del curso Complejidad Algorítmica tiene como objetivo desarrollar un videojuego basado en el juego de mesa Ubongo. Para esto, se tiene que considerar los temas aprendidos en clase, programación orientada a objetos (POO) y limitarse a usar el lenguaje de programación Python.

El Student Outcome de Ciencias de la Computación se define de la siguiente manera:

ABET – EAC - Student Outcome 1: Reconocer las responsabilidades profesionales y tomar decisiones informadas sobre prácticas de computación basadas en principios legales y éticos.

ABET – CAC - Student Outcome 2: Capacidad para interpretar, representar, comunicar y utilizar información cuantitativa diversa en situaciones de contexto real. Incluye calcular, razonar, emitir juicios y tomar decisiones con base en esta información cuantitativa.

El contenido del curso ha permitido aumentar la gama de conocimiento sobre algoritmos y estructuras de datos que tienen muchas utilidades en el contexto de la vida real, además de entender las diferencias entre estos y cómo es necesario favorecer ciertos algoritmos considerando el contexto y restricciones del problema, y el tiempo de ejecución de este. Uno de los aspectos más importantes del curso ha sido determinar la complejidad de estos diversos algoritmos que pueden solucionar cierto problema, pero no tienen el mismo tiempo de ejecución. De esta forma, se suele favorecer el algoritmo con menor complejidad a menos que el contexto y variables del problema obstruyan el proceso de este o las restricciones favorezcan a otro algoritmo, por lo que se opta por algoritmos que tienen mayor complejidad, pero son más eficientes en el contexto. Entre estas relaciones se puede encontrar Recursividad (Backtracking, Dividir y Conquistar, Fuerza Bruta) y Programación Dinámica, y los algoritmos de búsqueda de camino más corto en grafos con pesos Dijkstra, Bellmand-Ford y Floyd-Warshall.

Con esto, es posible tomar decisiones de acuerdo al SO1, e interpretar y producir información cuantitativa en el contexto real de acuerdo al SO2.

Por otro lado, el trabajo final era un problema de cómo usar de la forma más eficiente las funciones que proveían los módulos de Python, ya que no se podía observar una situación en la cual sean necesarios grafos o programación dinámica. Se implementó backtracking para encontrar soluciones a los rompecabezas del juego y que el jugador computadora los resuelva de esa forma, pero la complejidad es muy elevada (puede llegar a O(n4)), lo que ocasiona congelamientos en el juego cuando corre esta función. Se intentó usar dividir y conquistar para Merge sort, pero los elementos en el arreglo solo eran 4 y Merge sort no sería muy diferente a un Exchange sort, el cual sería codificado en menos líneas. Por estos motivos, de acuerdo a lo aprendido en clase, solo se aplicó fuerza bruta y backtracking, además de la forma de determinar la complejidad de los algoritmos. Cabe añadir que también se aprendió a usar módulos Python como Pygame y Numpy, los cuales son la base de la codificación del juego. Sin estos dos módulos, habrían existido más dificultades en el desarrollo del juego.

En general, se reconoce la responsabilidad que hay que tener al desarrollar un juego que sea amigable al usuario y código organizado, lo que cumple con el SO1. Además, es posible interpretar la complejidad que ciertas funciones tienen en el juego para determinar que parte del juego pesa más, lo que cumple con el SO1.

Para construir el juego, se ha tomado como base el trabajo parcial el cual había sido desarrollado usando solo el módulo PyQT5 5.14.2.2 y la mecánica de armar el rompecabezas no funcionaba. Había muchos botones y los modelos de las piezas eran confusos. El tablero se sentía anticuado y la interacción entre pantallas limitada. En general, solo se cumplían algunas mecánicas y ver las instrucciones, pero no era un juego que se pueda jugar por mucho tiempo. Por este motivo se empezó el diseño desde 0, se arreglaron todas las mecánicas, se agregaron más interacciones entre pantallas e incluso música de fondo, de forma que el jugador vea que el juego es entretenido. Este sigue siendo un primer proyecto con pocos meses de experiencia en Python, por lo que no es perfecto y puede ser pulido en varios aspectos, pero el resultado ha llegado a ser lo suficientemente satisfactorio para compartirlo, presentarlo, y jugarlo.

El presente documento va a primero listar los materiales utilizados para desarrollar el juego. Después, se explicará el desarrollo del proyecto haciendo comparaciones con el trabajo parcial debido a la relevancia de los cambios que existen entre ambos trabajos. Se hará mención de algunas de las actividades más relevantes y que impactaron más en el aprendizaje del lenguaje Python. Tras explicar el desarrollo, se presentará el diagrama de clases relacional. Las clases con sus atributos y funciones serán listadas por separado debido a que, entre estas, existen clases con muchas funciones que agrandarían la tabla en el diagrama, lo que ocasiona que sea menos comprensible. Luego, se mostrarán imágenes de funciones relevantes del juego y del juego mismo. Tras esto, se encuentra un manual del usuario con imágenes del juego y algunas indicaciones sobre qué hace cada botón, cómo jugar y progresar en el juego. Finalmente, se dará la conclusión del proyecto, sugerencias sobre el mismo y las referencias bibliográficas.

# MATERIALES

* Python 3.8
* Módulos de Python 3.8:
  + Módulo Pygame 1.9.6
  + Módulo Numpy 1.19
  + Módulo Pyinstaller 3.6
* PyCharm IDE 2019.3.4
* Aseprite 1.2.19

# DESARROLLO

* Tras determinar que una de las razones por las que el Trabajo Parcial no pudo ser culminado fue la falta de tiempo, este trabajo se empezó con 5 semanas de anticipación. En este tiempo, se cambió el framework de PyQT5 a Pygame debido a que tiene funciones que son más convenientes en la organización y codificación de funciones sobre el juego. También se implementó el módulo Numpy debido a su especialidad en manipular matrices, con funciones que permiten rotar, voltear, cambiar columnas, inicializar, mezclar la matriz. Esta vez no se usaron imágenes libres de la web. Todos los dibujos, fondos, fichas, tableros, tarjetas de jugador, números, letras, palabras, etc, fueron dibujadas desde 0 con la aplicación Aseprite. Aseprite es una herramienta de pixel art que es normalmente usada para crear sprites de personajes, fondos, terreno, entre otros. Se optó por esta herramienta de dibujo debido a que es fácil de usar, tiene una manipulación de gama de colores conveniente y es muy probable que la siga usando para futuros proyectos. También se optó por agregar música de fondo que combine con el estilo del dibujo, además de que estimula la atmósfera del juego en el usuario se vea enganchado en jugar más partidas y le da valor agregado.

Debido a que ya se tenía una base (Trabajo Parcial), se replicaron todas las funciones en el Trabajo Final excepto las 2 que utilizan los recursos del módulo gráfico: main y Application. En el Trabajo Parcial, estos archivos tenían implementado varias funciones de PyQT5, por lo que se tuvo que reconstruir todo el código con Pygame. Se descubrieron nuevas funciones que sirven para detectar la posición del mouse, detectar si se presiona una tecla y detectar si se hace un click.

Una de las ventajas de PyQT5 es que ya existe un efecto sobre los botones cuando el mouse pasa encima y cuando se hace click. Para recrear este evento, se crearon valores booleanos sobre cada botón que detecta si el mouse está encima. Si devolvía verdadero, aparecía un marco blanco alrededor del botón que indicaba que puedes hacer click. Este comportamiento se replicó en todos los botones del juego, por lo que resultó ser una característica importante del programa.

La actividad más tediosa fue obtener todos los rompecabezas con sus respectivas 6 soluciones para cada tiro del dado. Lamentablemente no se tiene el juego a la mano y no se encontraron fuentes que tengan las fichas listadas. Lo mejor que se pudo encontrar fue un video donde juegan 3 personas las 9 rondas, por lo que se pudieron obtener 27 tableros de los 36. Los últimos 9 tuvieron que ser duplicados de los ya existentes para que el juego puedo funcionar adecuadamente. Sin embargo, al copiar los datos, se encontró que algunos fueron copiados de forma errónea, por lo que, al terminar de construir el juego, se realizó un control de calidad en el cual se recorrían todos los rompecabezas y sus piezas para arreglar los errores si es que existen.

La actividad más difícil de desarrollar fue el evento en el que el jugador tiene que armar el rompecabezas. Todas las mecánicas principales funcionaban correctamente excepto esta. Tomé un descanso de 2 semanas sobre el proyecto para realizar tareas de otros cursos y pensar cómo codificar este evento. Finalmente se optó por el siguiente comportamiento: El jugador selecciona una pieza y se activa un booleano que indica que el rompecabezas está siendo armado. Internamente, se simula la pieza seleccionada como una matriz. Dependiendo de la posición y si el espacio está lleno, el color de la pieza será completamente azul o tendrá partes rojas. Si el espacio está libre, se puede hacer click y la matriz de la pieza será sumada a la matriz del rompecabezas, finalmente apagando el booleano del rompecabezas armándose. Numpy fue muy útil para algunas funciones de la matriz pieza, como la rotación hacia la izquierda o derecha y voltearla verticalmente u horizontalmente. Los controles resultaron ser muy sensibles, por lo que el jugador debe tener cuidado de no sacar el mouse del rompecabezas mientras el booleano está activado o este se desactivará deseleccionando la pieza. Cuando se determina que el rompecabezas está completo, las celdas donde hay piezas se vuelven azules y se avanza en el tiempo hasta que el cronómetro esté en 0.

Una actividad que resultó demorar más de lo que debería fue convertir el archivo principal, main, en un ejecutable. Al probar el ejecutable que se creaba correctamente, no encontraba un archivo llamada “libmpg123.dll”. Al final se encontró que solo se tenía que cambiar la versión de Python de 32 bits a 64 bits.

Una actividad de valor agregado vendría a ser el tiempo en el que los jugadores computadora completan el rompecabezas. Se generan 4 valores aleatorios entre 100 y el tiempo máximo menos 100 (el tiempo puede ser hasta 1000 ticks donde 1000 ticks = 60 s) y cuando el cronómetro llega a ese número, se elimina el número. Cuando ya no hay valores, el jugador máquina completó el rompecabezas. Como son aleatorios, el juego depende un poco de la suerte por si al jugador humano le toca un rompecabezas difícil o los jugadores máquina generan números que se eliminan rápido.

La actividad que está más relacionada a lo aprendido en el curso es la implementación del backtracking para hallar la solución del tablero del jugador computadora. Al principio quería evadir implementar esta función debido a que la complejidad es muy elevada. La función tiene que recorrer una matriz 6x6 para hallar la posición de las piezas en la matriz. Debe repetir este proceso 8 veces debido a que las piezas pueden tener hasta 8 posiciones diferentes (rotación en 0°, 90°, 180° y 270°, además de voltearla y crear 4 nuevos patrones con estas 4 rotaciones). Luego, se tiene que probar cada posición que puede tener esta pieza en el tablero, es decir, en el peor de los casos, se deben recorrer 6x6 posiciones. Si se encuentra una posición válida, se debe repetir este proceso con las otras 3 piezas teniendo la 1ra pieza añadida al tablero. Si las posiciones de la pieza son p, la matriz es n2, y las piezas son m, además de que el recorrido es recursivo, la complejidad, en términos de Big O, llegaría a ser O((p \* n2)m). Sin embargo, a pesar que esto es capaz de congelar el juego por algunos segundos, permite dar a entender que se aprendieron los algoritmos del curso y propone valor agregado al juego donde el jugador puede ver cómo la máquina va resolviendo sus puzzles.

En general, el Trabajo Final es un proyecto mucho más elaborado que el parcial debido a que ya se tenía una base de lo que se iba a trabajar, se revisaron los errores para no volver a cometerlos y mejorarlos, y se obtuvo un juego lo suficientemente satisfactorio para entregarlo.

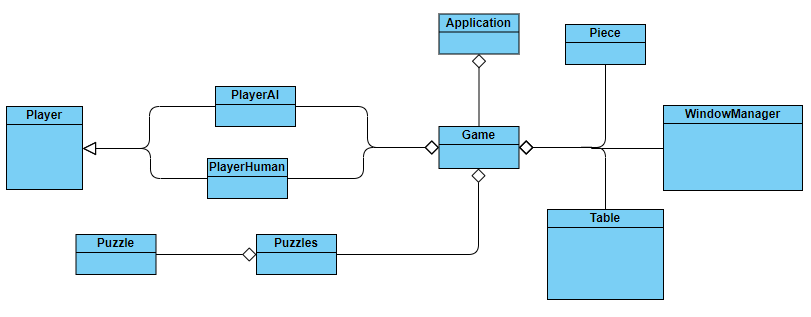
* **Diagrama de Clase**

Imagen 1. Diagrama de Clases relacional sin variables ni funciones

\*Las funciones y atributos son varios por lo que se limitó a mostrar las relaciones en el diagrama.

* **Clases**

**Application**:

*(atributos)*

* disp
* window
* game
* program\_running
* phase
* paused\_phase
* hints\_hidden
* current\_turn
* col
* making\_puzzle
* tp\_mouse
* selected\_piece
* selected\_piece\_id
* turn\_queue
* puzzle
* puzzle\_ai
* completed\_puzzle
* play\_hover
* instructions\_hover
* config\_hover
* exit\_hover
* return\_hover
* space\_hover
* begin\_turn\_hover
* hints\_hover
* piece\_hover
* blocked\_piece
* new\_game\_hover
* back\_menu\_hover
* minus\_time\_hover
* plus\_time\_hover
* minus\_turns\_hover
* plus\_turns\_hover
* minus\_cpu\_hover
* plus\_cpu\_hover

*(funciones)*

* Run()
* loadGame()
* runMenu()
* runGame()
* runInstructions()
* runConfig()
* mouseHovering()
* changeUsers()
* changeTurns()
* changeTime()
* showUserCards()
* showGems()
* showUsers()
* showUserGems()
* showUserPlaces()
* showPuzzles()
* showUnknownPieces()
* showPieces()
* showTimer()
* makingPuzzle()
* Combine()
* fixMousePos()
* aiTurn()
* updateAiBoard()

**Game:**

*(atributos)*

* player\_ai
* player\_human
* table
* puzle
* piece
* window
* turn
* time
* dice
* winner
* set\_time
* set\_turn

*(funciones)*

* startGame()
* Window()
* nPlayers()
* playerTurnPlace()
* gotoMenu()
* gotoGame()
* gotoInstructions()
* gotoConfig()
* throwDice()
* resetTimer()
* resetTurn()
* resetPlaces()
* resetUserTurns()
* resetPos()
* resetGems()
* startTimer()
* resetTable()
* resetPuzzle()
* chooseRow()
* updatePlaces()
* isAIPlaying()
* setAiPieceTimes()
* removeAiPieceTime()
* getPuzzles()
* getPiece()
* getPieces()
* refreshPuzzles()
* getWinner()

**Piece:**

*(atributos)*

* pieces

*(funciones)*

* createPieces()

**Player:**

*(atributos)*

* gems
* piece\_time
* pos
* turn\_place
* game\_place
* playing
* sols
* pieces\_on
* solved
* solving\_puzzle

*(funciones)*

* resetGems()

**PlayerAI:**

*(funciones)*

* generateInitialPos()
* chooseRow()
* setPieceTimes()
* removePieceTime()
* getSol()
* recursiveSol()

**PlayerHuman:**

*(funciones)*

* chooseRow()

**Puzzle:**

*(atributos)*

* form
* p\_pieces

**Puzzles:**

*(atributos)*

* puzzles

*(funciones)*

* createPuzzles()
* shufflePuzzles()

**Table:**

*(atributos)*

* gems\_pos
* n\_players

*(funciones)*

* generateTable()

**WindowManager:**

*(atributos)*

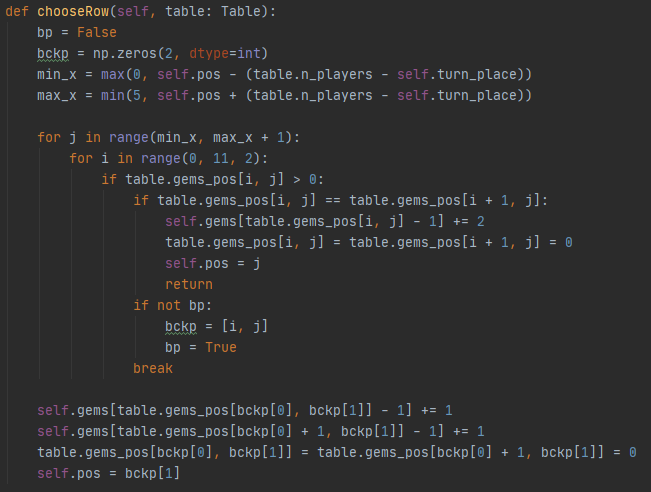
* window\_Type
* **Imágenes del Código**

Imagen 2. Función del jugador computadora para elegir una columna con 2 gemas del mismo color; sino elige la primera que encontró que tenga gemas

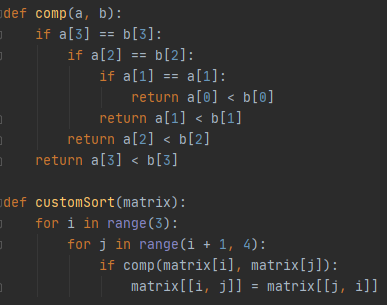


Imagen 3. Sort implementado para el puesto de los jugadores y el comparador para ordenar

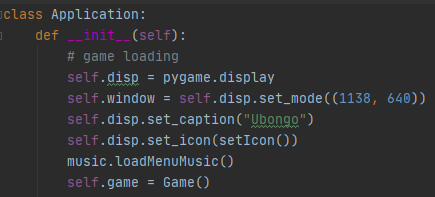


Imagen 4. Nueva forma de inicializar la pantalla del juego



Imagen 5. Función de Application con loop principal que permite que el juego corra mientras está abierto

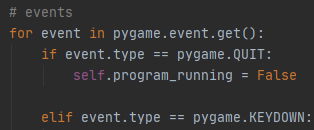




Imagen 6 y 7. Eventos de entrada que detecta el programa

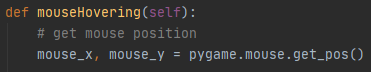


Imagen 8. Función para determinar si el mouse está encima de un botón o sobre algún objeto que le pueda hacer click

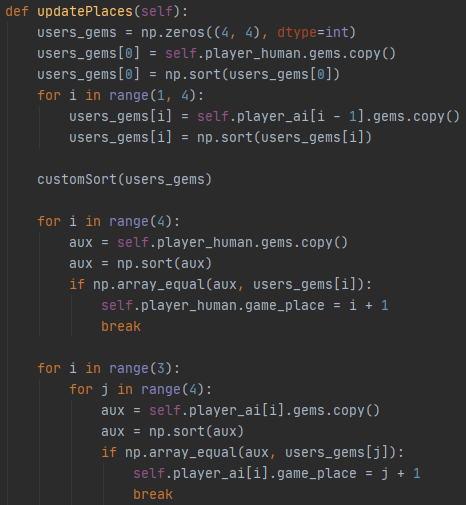


Imagen 9. Función que utiliza la función de Sort para ordenar los puestos de los jugadores dependiendo de cuántas gemas tenga cada uno

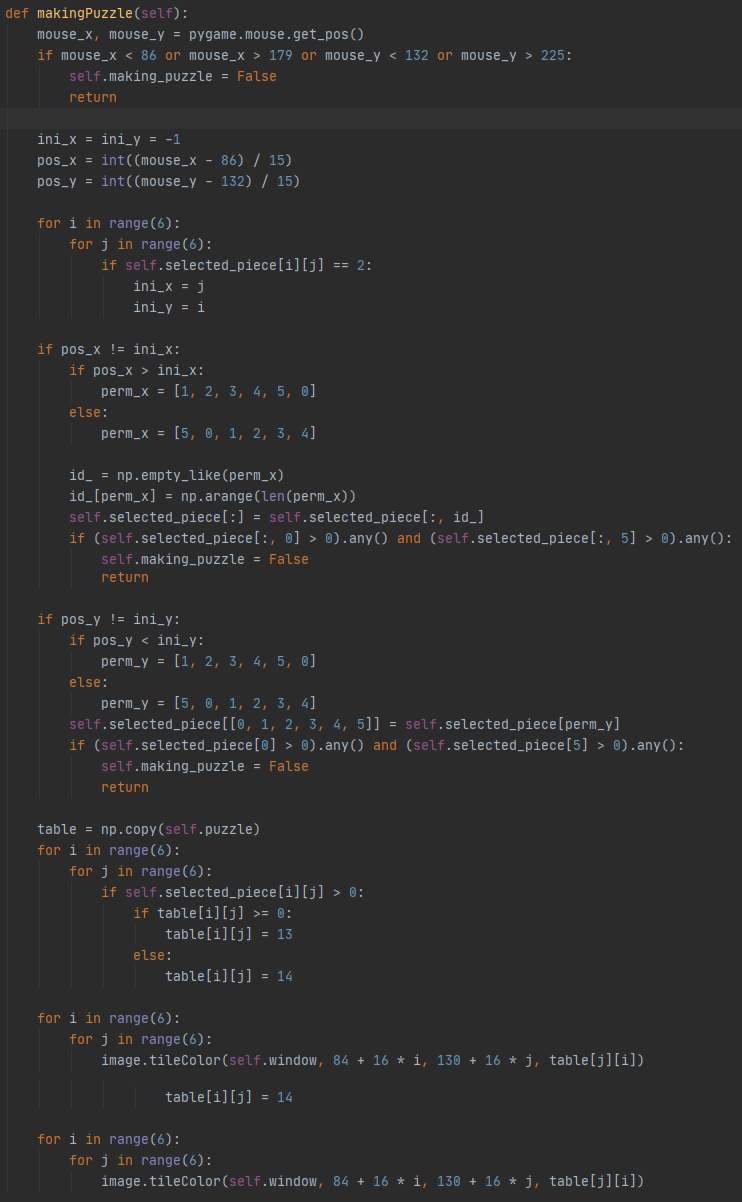
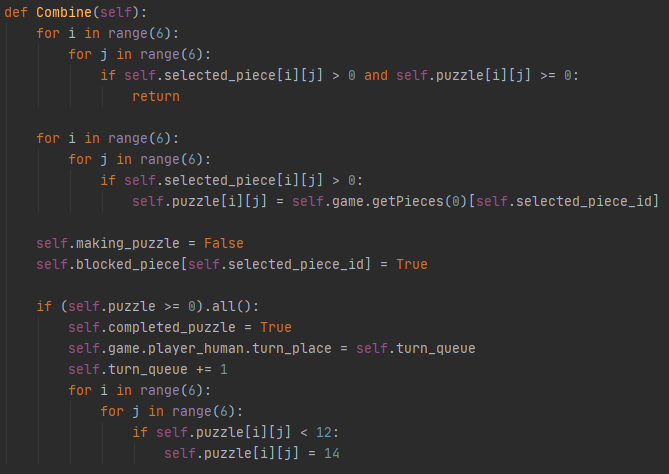


Imagen 10. Función que permite mover la pieza en el tablero y determinar si el espacio está disponible

Imagen 11. Función para colocar la pieza en el tablero y detectar si el rompecabezas está completo

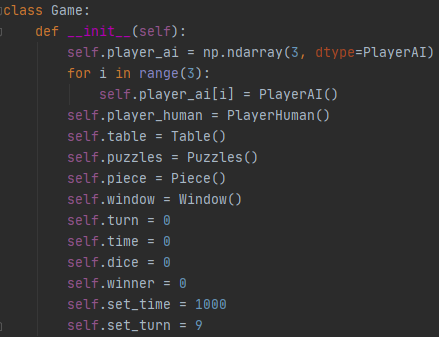
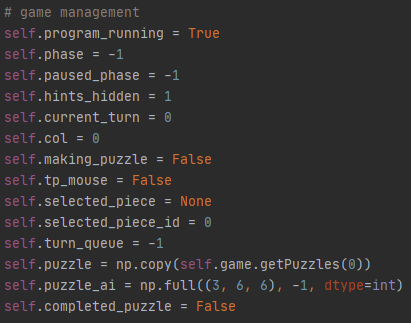


Imagen 12. Atributos de la clase Game



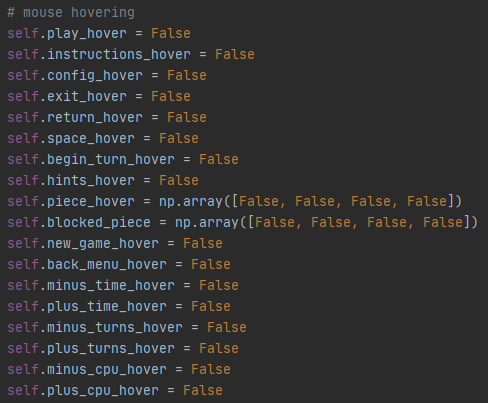


Imagen 13 y 14. Atributos de la clase Application

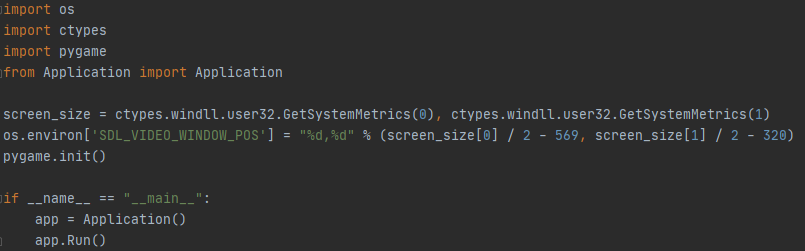


Imagen 15. Nuevo código del archivo main para que corra el programa

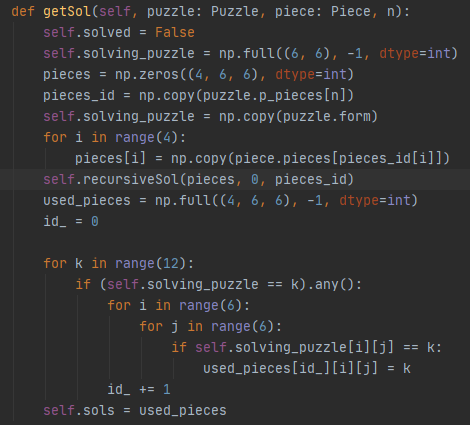
****

Imagen 16. Función para obtener la solución del puzzle del jugador máquina



Imagen 17. Función recursiva para hallar la posición de cada pieza en el puzzle del jugador máquina

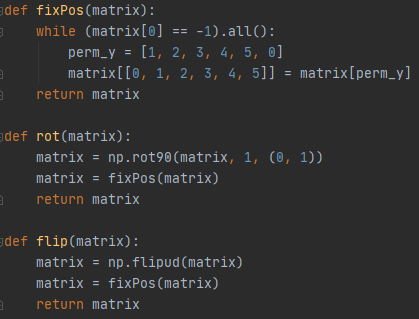


Imagen 18. Funciones para permutar la posición de las piezas en la recursión

* **Imágenes del Juego**

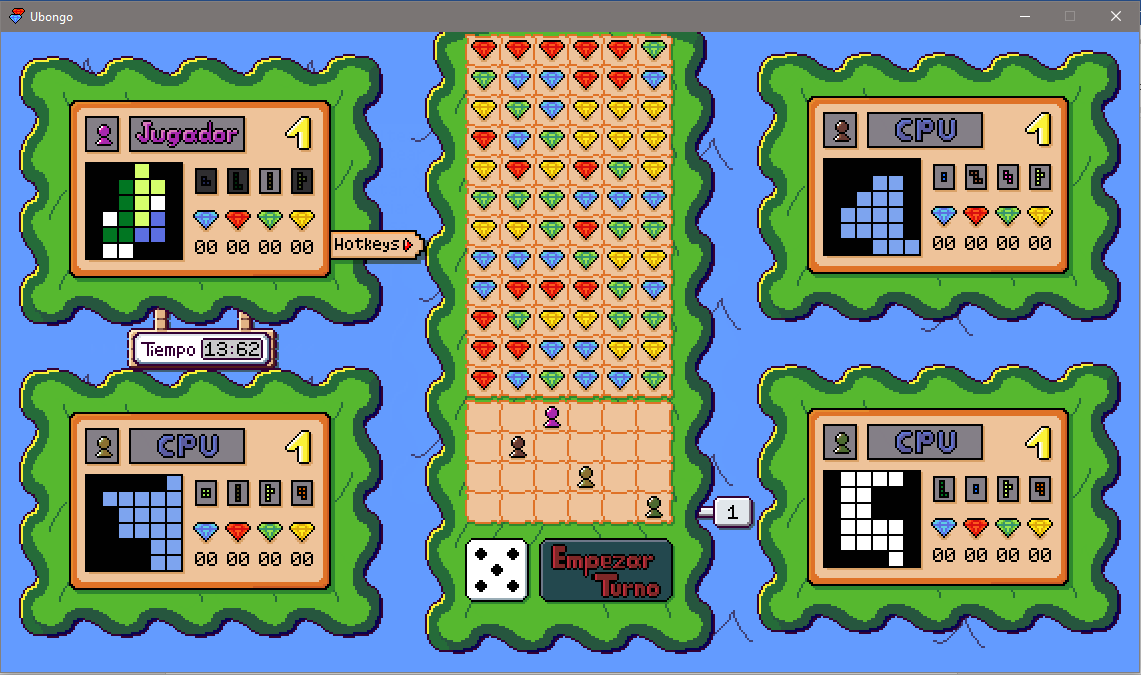
Imagen 19. Menú principal del juego

Imagen 20. Juego en curso con rompecabezas complejo

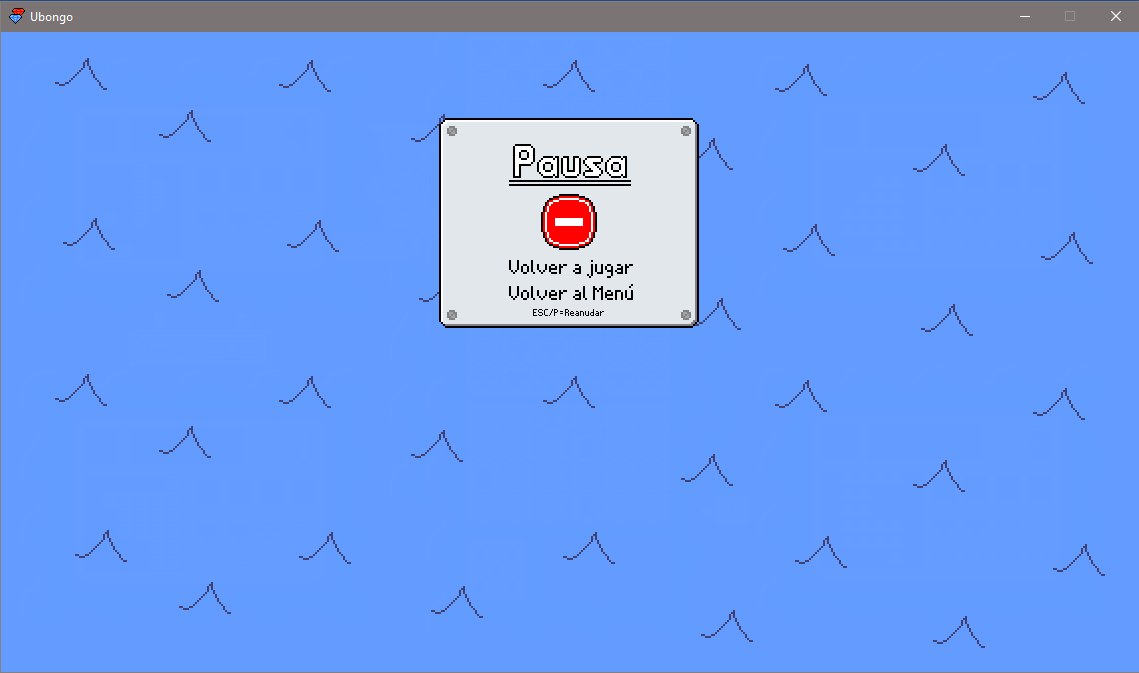


Imagen 21. Juego pausado

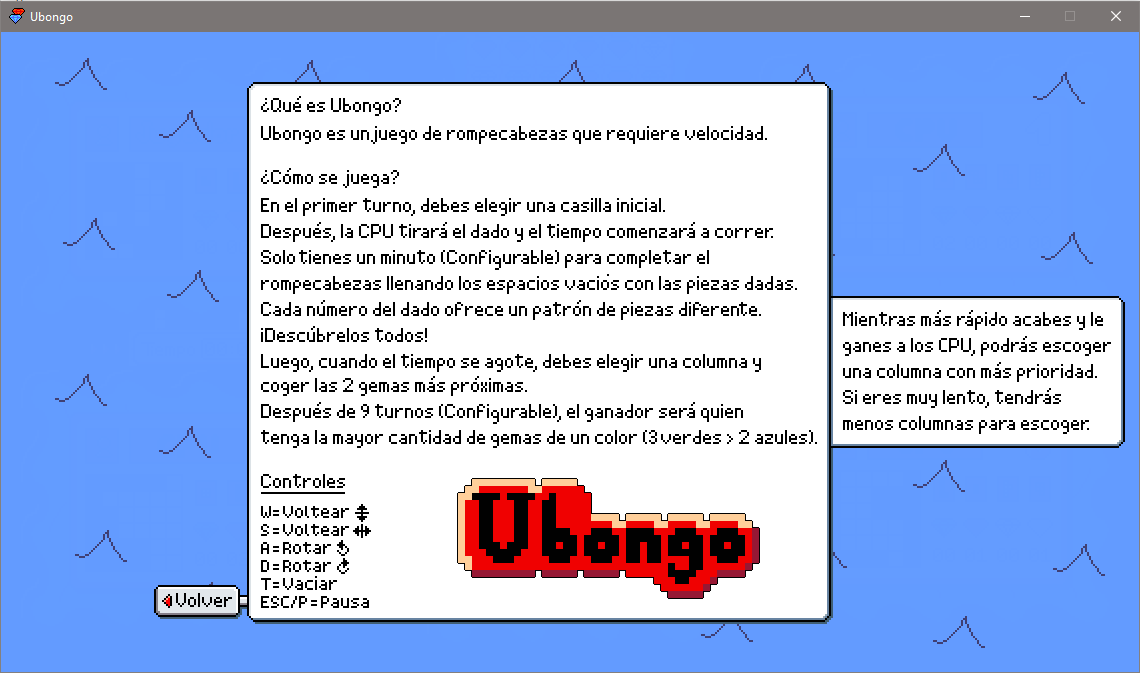
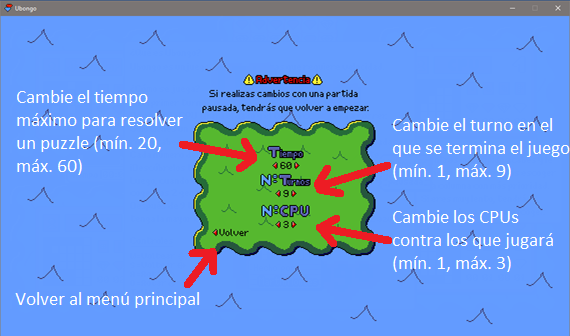
Imagen 22. Instrucciones del juego además de los controles

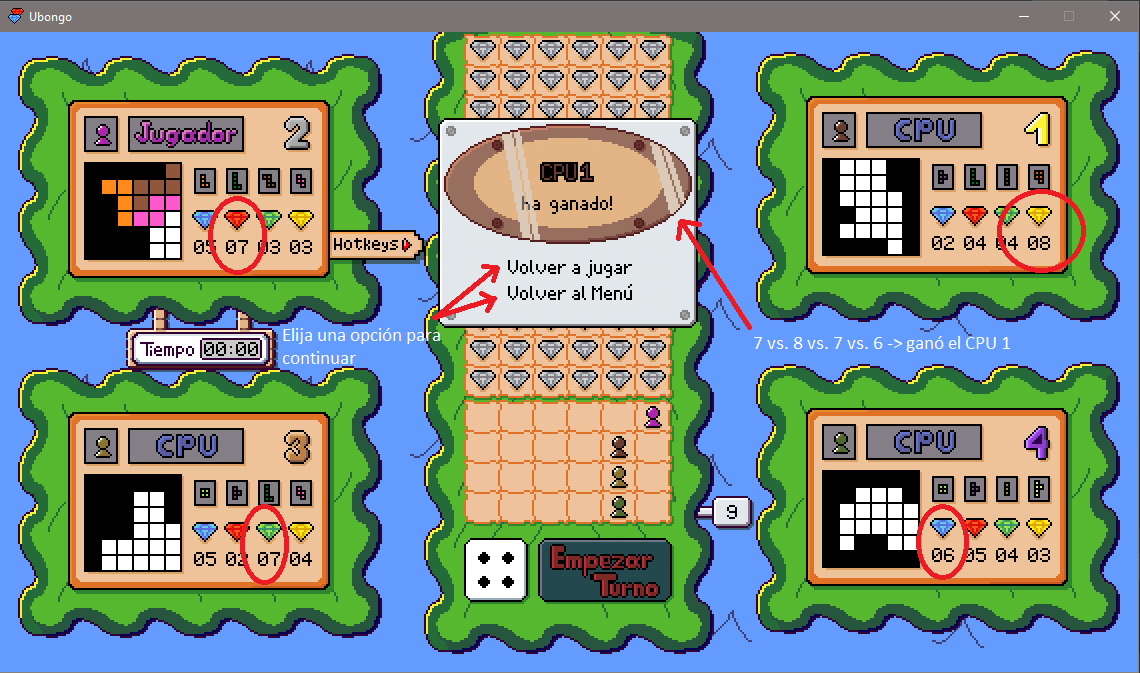
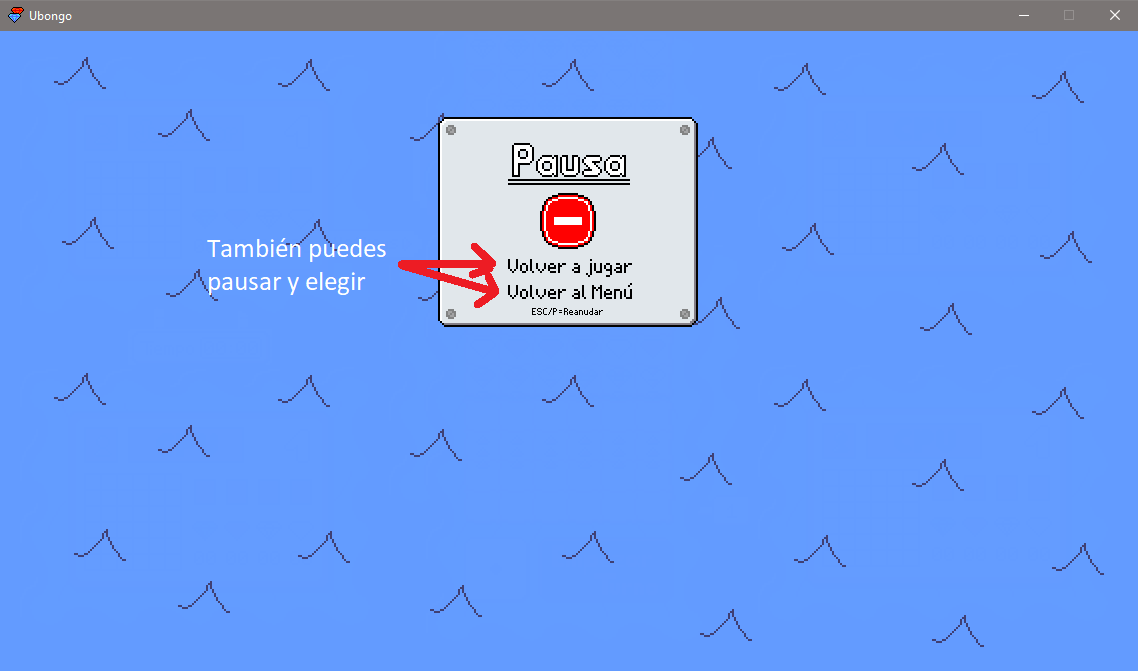
Imagen 23. Configuración de algunos elementos del juego



Imagen 24. Pantalla de victoria tras una partida blitz contra 1 jugador máquina

# MANUAL DE USUARIO





**Video de un gameplay del juego:**

<https://youtu.be/YiZiu7cNuDg>

# CONCLUSIONES

* Con respecto a lo aprendido en el curso, se ha podido implementar todo lo posible que pueda beneficiar al juego, lo que incluye fuerza bruta y backtracking. Programación dinámica y grafos no fueron necesarios debido a la naturaleza de rompecabezas del juego, es decir, no se encontraron formas de implementar estos algoritmos en el juego.
* Con respecto a los gráficos, se ha obtenido mayor satisfacción que en trabajo parcial y se cumplió el objetivo. Pasó de un estilo anticuado a uno de pixel art, además de que se aprendió a usar una herramienta para edición de pixel art que será de utilidad en el futuro. Los botones proveen retroalimentación al pasar por encima de ellos, los dibujos son más coloridos que los del trabajo parcial e incluso se pudo añadir música de fondo.
* Con respecto a la jugabilidad, se puede decir que también se cumplió con el objetivo. A diferencia del parcial, no solo se completaron las mecánicas que faltaban, también se añadieron más que enriquecen el contenido del juego, incluyendo configuración de turnos, tiempo y cantidad de jugadores máquina, música de fondo, el jugador máquina resolviendo el puzzle, controles para mover las piezas más amigables, etc.
* Este proyecto permitió conocer con mayor detalle el módulo Numpy, el cual era mencionado varias veces en temas como Machine Learning, lo cual será importante para futuros cursos. También se conoció el módulo Pygame, el cual fue más cómodo de usar que PyQT5 debido al contexto del proyecto. La forma en la que se desarrolló el proyecto permite tener una mejor idea de cómo elaborar un videojuego.
* Con respecto a los temas del curso, este proyecto permitió usar los algoritmos en contexto real y experimentar su utilidad al momento de programar.

# SUGERENCIAS

* A pesar que se han cumplido los objetivos, se debe admitir que hay optimizaciones que se pueden implementar al juego. Por ejemplo, el movimiento de piezas es muy sensible y podría tener un mayor margen de error o reaparecer si el mouse vuelve dentro del tablero. También el backtracking podría ser reemplazado por soluciones preexistentes para reducir la complejidad, pero demoraría más en escribirse. Finalmente, se podrían agregar efectos de sonido para mayor retroalimentación sobre la interacción con los botones y las piezas, y entre el usuario y el juego en general.
* Como este proyecto no se vio posible implementar backtracking, un juego relacionado a laberintos sería ideal como nuevo proyecto. Si se quiere ver la solución en el laberinto, se usaría la función del grafo para hallar el camino. Este concepto es muy general, y se le podría agregar, por ejemplo, más niveles, un contrarreloj, vs CPU, adquisición de puntos mientras más rápido se resuelve, etc.

# BIBLIOGRAFÍA

Visiados Mesa .(2016). *Ubongo (Nueva Edición) - Juego de mesa – Gameplay.* Recuperado de: <https://www.youtube.com/watch?v=bHT9E8Tk3mw> [1 de mayo del 2020]