

Universidad Simón Bolívar

Laboratorio de Algoritmos y Estructuras 1

Prof. Jean Carlos Guzmán

**Desarrollo de Solitario Chess para el Laboratorio de Algoritmos y Estructuras 1**

Realizado por:

* César Colina 13-10299
* Orlando Chaparro 12-11499

Caracas, Diciembre 2016

**ÍNDICE**

Introducción................................................................................................................ Pag 3

Diseño......................................................................................................................... Pag 4

Estado actual del proyecto.........................................................……………………………….Pag 8

Conclusión..................................................................................……………………………….Pag 9

Bibliografía.................................................................................................................. Pag 10

**INTRODUCCIÓN**

Solitario Chess es un rompecabezas de lógica e ingenio para un solo jugador basado en las reglas de ajedrez tradicional, este informe tiene como objetivo explicar detalladamente cómo fue el desarrollo e implementación del código para el buen funcionamiento del programa. El objetivo del proyecto es desarrollar “desde cero” un juego de Solitario Chess totalmente funcional y con las siguientes características: 4 niveles de dificultad (Fácil, Difícil, Muy Difícil y entrenamiento) las cuáles pueden cargarse desde cartas desafío predeterminadas o carga desde el teclado (usuario). El programa debe poder empezar una partida nueva, cargar una partida guardada anteriormente, mostrar la tabla de records y finalmente salir del juego. Además el juego guarda y muestra en pantalla el nombre del usuario y el tiempo restante para acabar la partida. Luego, mientras el usuario se encuentra jugando, debe tener las posibilidades de pausar la partida, deshacer jugada, posible jugada (solución) y terminar partida, acotando que dichas funcionalidades dependen del nivel de dificultad. Para la realización de todas estas especificaciones hicimos uso del análisis descendente por medio del desarrollo de diversas funciones que realizan cada una un fragmento del problema distinto. Con dicho informe el lector podrá tener un panorama general de cómo desarrollar desde el inicio un juego de Solitario Chess totalmente funcional desarrollado con la librería de Pygame en el lenguaje de programación Python3. Finalmente el informe se divide en portada, índice, introducción, diseño, estado actual del proyecto, conclusiones y bibliografía.

**DISEÑO**

Para definir la solución general del programa decidimos crear funciones que solucionaban problemas más sencillos de resolver que mediante su composición lográramos tener la solución general del programa. Estas tareas se dividieron a su vez en sub-tareas las cuáles fueron resueltas mediante sub-programas.

Primeramente es bueno saber que nuestra estructura de dato principal en el juego es una lista, la cual representa nuestro tablero con sus respectivas posiciones (a1,b4,c2,etc...). Para mayor comodidad a dicha lista la llamaremos de ahora en adelante TABLERO. Esta lista posee como elementos 4 listas que representan las 4 filas del tablero y los elementos de estas listas representan las columnas del tablero. Es decir TABLERO[0] representa la fila 4, TABLERO[1] representan la fila 3 y así sucesivamente. Ahora bien TABLERO[0][0] representa la fila 4 y la columna “a”, luego TABLERO[0][1] representa la fila 4 y la columna “b” y así sucesivamente. Finalmente TABLERO representa las posiciones libres u ocupadas en nuestro tablero de juego.

Empezamos a desarrollar una función AbrirArchivo() bastante sencilla, la cual abre un archivo.txt donde se encuentra las cartas desafío, es decir las fichas y sus posiciones para luego pasar dichos datos a una lista mediante métodos como split() ,append() y readlines(). Esta lista ahora posee los datos de las fichas y sus posiciones.

Luego creamos otra función LlenarTablero() que a partir de la lista creada anteriormente es capaz de traducir estos datos y usarlos para llenar nuestro TABLERO y así saber que fichas hay en juego.

Posteriormente creamos una función denominada selecciondePieza() que le pregunta constantemente al usuario que ficha desea mover, dicha función posee sus aserciones para comprobar que la ficha exista. Esta función nos arroja unas coordenadas que nos informa en qué posición se encuentra la ficha según nuestro TABLERO y qué tipo de pieza es la ficha que el usuario desea mover.

Con dichas coordenadas y sabiendo que tipo de pieza queremos mover, pasamos estos datos a nuestra función principal de movimiento ValidarJugada(), la cual se encarga, dependiendo de qué tipo de pieza estamos seleccionando, calcular las posibles jugadas que tiene determinada pieza y comparar dichas posibilidades con las que el usuario desea ingresar.

Como es una de las funciones principales, explicaremos detalladamente como validamos las jugadas para el peón.

Para el peón: Como el movimiento del peón es solo hacia arriba y en diagonal(una casilla), colocamos la siguiente condición: Si la coordenada nueva de las filas es igual a la coordenada vieja de las filas – 1, y (la coordenada nueva de las columnas es igual a la coordenada vieja de las columnas +1 o la coordenada nueva de las columnas es igual a la coordenada vieja de las columnas -1) entonces es una jugada válida y procedemos a actualizar TABLERO, moviendo de la coordenada vieja a la coordenada nueva nuestra pieza y removiendo cualquier pieza que estaba anteriormente en la coordenada nueva.

Finalmente con la unión de estas funciones podemos empezar a jugar, resolviendo así el problema principal del juego. Para la unión de dichas funciones nos valimos de nuestra función principal Iniciar(), la cual carga e inicia todos los componentes principales del programa mediante un bucle infinito que constantemente está llamando a las funciones esperando la interacción del usuario.

Claro está que para ver el juego en una interfaz utilizamos otras funciones secundarias que recibiendo los datos de TABLERO son capaces de dibujar en pantalla las imágenes de acuerdo a su posición.

El otro gran problema del programa, es saber cuando el usuario gano o perdió. Para ello aplicaremos el mismo análisis descendente hecho anteriormente aplicando la técnica de dividir el problema en problemas más fáciles de resolver.

Creamos una función lugaresDondeHayPiezas(), que recibe como parámetro TABLERO, que mediante un ciclo for es capaz de identificar las piezas que hay y en qué posición están. Éstos datos serán agregados mediante el método append() a dos listas nuevas que serán nuevamente nuestra estructura de datos principal. Finalmente poseemos una lista con todas las piezas y otra lista con sus posiciones correspondientes en TABLERO.

Luego mediante una función llamada peonJugadas(), le pasamos como parámetros las coordenadas donde se encuentra el peón y mediante un algoritmo parecido a ValidarJugada() y un ciclo for es capaz de calcular todas las posibles jugadas del peón y agregarlas a una lista mediante el método append(). Finalmente creamos una función para cada tipo de pieza (caballoJugadas(),alfilJugadas(),etc).

Ya para finalizar y resolver el segundo gran problema (saber si el usuario gana o pierde) creamos una función evaluacionDeJugadas(), que recibe como parámetro las dos listas resultantes de lugaresDondeHayPiezas() e identificando que tipo de pieza y en que coordenada esta la pieza, llama puntualmente a la función necesaria para calcular todas las posibles jugadas de la pieza según su coordenada, por ejemplo peonJugadas(). Así esta última función nos retorna una lista con todas las posibles jugadas de todas las piezas en el juego según su coordenada y si dicha lista no posee elementos y la cantidad de piezas en TABLERO es mayor a 1, es porque el usuario ha perdido. Luego si la cantidad de elementos de la lista de todas las posibles jugadas es igual a 0 y la cantidad de piezas en tablero es 1 entonces el jugador gano la partida.

Con todo lo descrito anteriormente hemos explicado cómo resolver los dos grandes problemas principales en el programa, es decir, resolver los movimientos de las piezas y las validaciones de las mismas, y que el programa reconozca cuando el jugador ganó o perdió. Ahora describiremos brevemente otros aspectos que también conforman el juego pero desde una perspectiva más general y desde el punto de vista del usuario.

**IDENTIFICACIÓN**

Para el comienzo del programa, el usuario debe identificarse con su nombre, lo cual se resolvió con el reconocimiento de “eventos” en pygame mediante el teclado, guardando en una variable dicho nombre. Luego el usuario presiona enter y accede al siguiente menú.

**PARTIDA NUEVA**

Si el usuario elige la opción de partida nueva, se abre un segundo menú con las diferentes dificultades con que se puede iniciar una partida (Fácil,Difícil,Muy Difícil,Entrenamiento). Para la transición de menús sencillamente utilizamos condicionales y dibujamos las imágenes en pantalla. Para todas las dificultades el usuario es capaz de elegir entre carga desde el teclado o cartas desafío predeterminadas. El manejo del tiempo en todas las dificultades (excepto en modo entrenamiento) fue gracias a los métodos de pygame tales como pygame.time.Clock() y pygame.time.get\_ticks()

**PARTIDA FÁCIL**

Si el usuario elige la opción de partida fácil, tiene un tiempo límite de 3 minutos y tiene la opción de deshacer jugada. La opción de deshacer jugada fue implementada guardando TABLERO en una variable auxiliar y así poder tener un respaldo de la última jugada. Si el usuario usaba la opción deshacer jugada sencillamente se imprime en pantalla el ultimo TABLERO guardado. Adicionalmente el usuario puede pausar la partida lo que ocasionaba que el tiempo dejara de contar.

**PARTIDA DIFÍCIL**

El usuario solo dispone de un minuto y medio para acabar la partida y puede pausar la partida.

**PARTIDA MUY DIFÍCIL**

El usuario solo dispone de 2 minutos para resolver 3 cartas de desafío. Al momento de que el jugador gana (ya explicamos como el algoritmo funciona según nuestro segundo gran problema) el programa carga otro archivo.txt y así hasta terminar la partida.

**PARTIDA DE ENTRENAMIENTO**

El usuario no está limitado por el tiempo y tiene la opción de SOLUCION, la cual según nuestro algoritmo es sencillamente una posible jugada (ya explicamos como el algoritmo funciona según nuestro primer gran problema) que es mostrada en pantalla.

**CARGAR PARTIDA**

Si el usuario elige cargar partida, es capaz de recuperar todo el TABLERO tal cual lo dejo la ultima vez, lo que hace nuestro programa es abrir un archivo.txt con todos los datos guardados, necesarios para dibujar nuevamente en pantalla la partida que se quiere cargar.

**MOSTRAR TABLA DE RECORDS**

El usuario es capaz de revisar los records guardados anteriormente, los cuales detallan cuantas cartas desafío resolvió y en cuanto tiempo. El programa tan solo debe guardar cuantos archivos.txt fueron abiertos y el tiempo en que culmino. Luego debe hacer una comparación con la actual tabla de records y actualizar la lista.

**SALIR DEL JUEGO**

La opción de salir del juego se implemento con el método sys.exit()

**ESTADO ACTUAL DEL PROYECTO**

El programa actualmente está totalmente funcional, empecemos detallando por Las dificultades.

El usuario puede iniciar una partida de cualquier dificultad:

-Si el usuario inicio una partida fácil tendrá un tiempo de 3 minutos máximo y tiene la opción deshacer jugada y de pausar la partida,

-Si el usuario inicia una partida difícil tiene un máximo de 1 minuto y medio y puede pausar la partida

-Si el usuario inicia una partida muy difícil tiene un máximo de 2 minutos pero deben ser resueltas 3 cartas desafío.

-Si el usuario inicia una partida de entrenamiento tiene la posibilidad de pedir una pista(posible solución).

El usuario es capaz de cargar una partida guardada anteriormente recuperando todos los datos exactos.

El usuario es capaz de ver la tabla de records que se actualiza cada vez que la partida termina.

El usuario es capaz de salir de la partida cuando desee.

El usuario es capaz de elegir entre una carta desafío predeterminada o su propia carta desafío ingresada por el teclado.

Durante el juego están totalmente funcional las opciones jugar,pausar,deshacer,terminar y solución.

El juego muestra en pantalla el nombre del usuario y el tiempo restante para que se acabe la partida.

**CONCLUSIONES**

El resultado de este proyecto fue grato debido a que el grupo de trabajo logró un juego totalmente funcional, cumpliendo con todos los requerimientos del proyecto. Adquirimos grandes conocimientos y pusimos en práctica técnicas aprendidas en el laboratorio durante el trimestre tales como el análisis descendente y y el método de “Divide y Vencerás”. Durante el desarrollo del programa tuvimos dificultades principalmente con el ingreso del texto y el dibujar en pantalla las imágenes(referente a la transición de escenas) pero finalmente pudimos resolver. La librería de Pygame nos hizo perder muchas horas debido a la inexperiencia con la misma, sin embargo después de días tratando de resolver las dificultades pudimos culminar el proyecto. Creemos que es vital para el buen aprendizaje en la materia dichos proyectos ya que nos aportan una gran cantidad de experiencia sin embargo exhortamos al mejor uso del tiempo en la asignación del mismo.

BIBLIOGRAFÍA

* Pygame. Pygame Front Page. <http://www.pygame.org/docs/>
* Sweigart A. (2012) Making Games with Python & Pygame.
* Código Facilito (2014) Pygame. <https://codigofacilito.com/cursos/pygame>
* Meza, O y Ravelo, J. (2012) Introduccion a la programacion. Recuperado de: https://moodle.asignaturas.usb.ve/pluginfile.php/44225/mod\_resource/content/1/IntroduccionALaProgramacion.pdf