Caracas, 14 de diciembre de 2016
Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2691 – Laboratorio de Algoritmos y Estructuras I
Trimestre Septiembre-Diciembre 2016

Informe
Proyecto de Laboratorio de Algoritmos y Estructuras I

Integrantes:

Gustavo Castellanos 14-10192. Prof. Jean Carlos Guzmán
Yuni Quintero 14-10880. Prof. Josué Ramírez

Introducción:

Solitaire Chess es un juego de lógica altamente relacionado con el clásico Ajedrez. En ambas versiones se utilizan las mismas piezas y reglas con ciertas excepciones: en Solitaire Chess participa un sólo jugador, el tablero tiene dimensiones 4x4 y cada movimiento que se realice debe ser una captura hasta que sólo quede una pieza en el tablero. El juego es interactivo y retador donde la meta es poder resolver los distintos tableros en el menor tiempo posible. El jugador debe ser capaz de considerar todos los posibles movimientos hasta lograr la solución mas óptima.

En el siguiente informe se explicará cómo se llevó a cabo este proyecto, describiendo las diversas estructuras de datos y funciones que se utilizaron y cómo éstas fueron fundamentales para la resolución del problema y la ejecución del juego, concluyendo así que éste funciona de forma correcta y esperada.

El proceso de resolución del problema está dividido en tres partes. Primero, la creación de la interfaz gráfica del juego que incluye animaciones, imágenes, figuras, cuadros de texto, etc. Segundo, definir todas las funciones que verifican la correctitud del juego, es decir, los movimientos de las piezas que el usuario introduce, el tablero que se ingresa, las opciones de Jugar, Pausar, Cargar Partida, Deshacer Jugada, entre otras, las cuales dependen del nivel en que se esté jugando (Entrenamiento, Fácil, Difícil y Muy Difícil). Además, se definen funciones que permiten al usuario interactuar con el juego mediante el teclado. Tercero y último, el ciclo principal del juego, donde las dos partes anteriores coinciden para unificar y ejecutar todo. El ciclo se va actualizando mientras el usuario realiza los cambios respectivos en el juego, es decir, las decisiones que toma, a cuál ventana se dirige, cuántas fichas han sido capturadas para declarar la partida ganada o perdida, entre otras.

Para la implementación del juego se utilizó la librería gráfica de python llamada Pygame. Entre los objetivos de este proyecto se encuentran:

-. Crear un ambiente divertido y a su vez sencillo y cómodo para el usuario y así disfrutar del juego. Las animaciones pueden ser una gran herramienta para lograr este objetivo. También, la utilización de colores y figuras correctas para crear una estética sencilla y calmada dada la seriedad y dificultad de Solitaire Chess, entre ellos: el beige o marrón claro, vinotinto, blanco y simples cuadros de texto con sus determinadas especificaciones.

-. Permitir que el usuario interactúe y sea independiente en el juego, es decir, pueda navegar entre ventanas y querer dirigirse a un determinado destino. Es de suma importancia que el jugador sepa cómo manejar el juego y no se sienta perdido, por ejemplo, no saber que tecla presionar para pausar el juego o cargar un tablero aleatorio. También es fundamental la creación de funciones que permitan la interacción del usuario con el juego y así éste podrá cargar su propio tablero, escoger el nivel de dificultad a jugar e incluso introducir su nombre.

-. El funcionamiento correcto del juego considerando las especificaciones para cada nivel, las reglas para los movimientos válidos en Solitaire Chess, las transiciones entre ventanas, etc. Para esto es necesario la utilización de distintas estructuras de datos y funciones que verifiquen esas acciones. Además, cumpliendo con todo lo que conlleva ser un “juego”, permitir al usuario poder ver un registro o un record de sus partidas pasadas y así poder superar más retos y tableros, haciendo que el juego sea interesante y competitivo consigo mismo.

Por último, en el informe también se expondrán las dificultades, prioridades y los pasos a tomar para resolver dichos problemas. A continuación se explicará cuáles fueron los subproblemas claves para la implementación del juego. siendo estos la creación de la interfaz gráfica, el desarrollo de cada nivel, habilitar el teclado para que el usuario interactúe con el juego y la creación de los distintos archivos .txt que contienen partidas guardadas, nuevas y/o aleatorias y el registro de los récords de los usuarios.

Diseño:

Al comenzar el desarrollo del juego, lo primero que se consideró fue crear las principales ventanas de la interfaz gráfica y la navegación entre ellas. Para ello se declaró una variable *window* la cual indica en que ventana del juego el usuario se encuentra; se definieron funciones para cada estado de *window* las cuales "imprimen" en pantalla la interfaz correspondiente a cada ventana, entre ellas, *MAIN* que se refiere al menú principal, *INGAME* siendo el menú de juego donde se visualiza el tablero, las fichas, el tiempo, etc, *START* que es la ventana de inicio de *Solitaire Chess*, donde el jugador ingresa su nombre de usuario, entre otras ventanas.

Una de las prioridades principales de la implementación del proyecto fue cómo permitir al usuario interactuar con el juego, el medio para esto siendo el teclado. Para poder registrar el nombre de usuario en la ventana *START* se realiza el llamado a la función *getUser* que modifica la variable *userName* segun el input/entrada del usuario; también valida que *userName* tenga un formato correcto, i.e, no sea un cadena de caracteres vacía y cuando el usuario oprime la tecla Enter, éste no debe ingresar espacios ni caracteres extraños (incluyendo letras pertenecientes) al abecedario español) y que no pueda borrar un caracter si el string es vacío. De forma similar, para que se procese el tablero que el usuario ingresa y los movimientos, se analiza en la función principal del juego qué tipo de eventos ocurren, es decir, si se ingresan en el teclado caracteres no válidos para el formato del tablero y movimiento, si la tecla ingresada fue *K_BACKSPACE (*es decir, borrar), entre otros. El string que el usuario ingresa como formato del tablero se guarda en la variable *chess* que luego se utilizará en *genBoard* para generar el tablero de juego y color las piezas, y el string que el usuario ingresa como movimiento se guarda en la variable *movement* que se utilizará posteriormente en la función *getMovement*. Cabe destacar que la mayoría de las interacciones con el usuario se procesan en la función principal del juego, *main*, cada vez que dentro de la guardia de un if verifica primero que tipo de evento ocurrió, es decir, que tecla fue presionada, y, depende cual ésta fue, ocurrirá una instrucción determinada, como por ejemplo, si el usuario se encuentra en la ventana LEAVE, la cual le pregunta al jugador si salir o no del juego, se procede a preguntar si el evento fue oprimir la tecla 1. Si salir o 2. No salir o 0 . Regresar a la partida, ya que la función próxima a ejecutarse depende de cuál tecla fue oprimida.

Para que el juego pudiese interpretar correctamente una configuración de tablero de la forma *Pieza1Posición1-Pieza2Posición2-…-PiezaNPosiciónN,* con $0 < N < 11$, se definió una función *getNumber* la cual recibe como parámetro un carácter y devuelve un entero que indicará el número de la columna correspondiente. Funciona como una biyección *f: {"a", "b", "c", "d"} → [4].*

En conjunto con la función *getNumber* se utiliza una matriz (lista de listas en python) *grid* 4x4 de caracteres, donde $grid[i][j] = Piezak,\ 0 <= i, j <= 4,\ 1 <= k <= N.$ Dicha matriz es utilizada por la función *printBoard* para mostrar las piezas según la configuración de tablero. La función *getNumber* se utiliza nuevamente para interpretar las jugadas del usuario. La función *genBoard* modifica la matriz *grid* inicializada con caracteres "Z" en todas sus casillas (para indicar que actualmente no se encuentran piezas en esas posiciones) mediante el string *chess* que contiene el tablero que ingresó el usuario por input y el método *split,* e inserta los caracteres P,D,R,A,T,C en las posiciones respectivas de la matriz donde no haya sido modificada anteriormente, esto permite que la función verifique que no se introduzca más de una pieza en una misma posición. Además, la función verifica la cantidad de piezas necesarias y suficientes para que el tablero cargado sea válido.

Mediante bloques de código *try: except:* se verifica que la configuración del tablero ingresada por el usuario cumple con las siguientes restricciones: hay a lo sumo un Rey y a lo sumo una Dama; hay a lo sumo dos ocurrencias de las piezas que no son ni Rey ni Dama; no hay dos piezas en una misma casilla y hay al menos mas de una pieza en el tablero.

Para el caso en el que el usuario escoge que el programa genere una configuración del tablero, se creó un archivo partidasnuevas.*txt* con una lista de configuraciones de tableros, el cual es accedido por la función *randBoard*. Ésta utiliza el método *readlines* para crear una lista con las líneas del archivo y luego, mediante el módulo *random* de python, escoge una línea al azar la cual corresponderá a la configuración del tablero a jugar.

Después de haber escogido el modo de carga del tablero para jugar, el usuario elige qué nivel desea jugar, entre los cuales se tiene Entrenamiento, Fácil, Difícil y Muy Difícil.

Para el nivel Entrenamiento, no existe un límite de tiempo para resolver el tablero, además, está habilitada la opción de Generar Solución que consiste en sugerir al usuario que movimiento puede ejecutar a partir de una posición inicial dada. Esta opción se ejecuta correctamente gracias a la función *genSolution*, que recibe como parámetro un string que contiene la posición de la pieza en el tablero en el formato "PiezaPosición" y verifica todas las posibles posiciones en las cuales la pieza inicial puede capturar a otra según sea válido el movimiento. También, se le permite al usuario Deshacer Jugadas, utilizando el método *deepcopy* el cual crea una copia exacta de la matriz *grid* y anexa esa copia a otra matriz llamada *states.* Gracias a este método, se crea una copia del estado del tablero antes de la ejecución de un movimiento y así poder deshacer la última jugada realizada.

Para el nivel Fácil la opción Deshacer también está habilitada, además, se le permite al jugador poder guardar una partida en sus condiciones actuales y poder reanudarla posteriormente cargándola desde un archivo *partidasguardadas.txt*. La función que ejecuta esta acción es *save,* la cual a partir de la matriz *grid,* que contiene el estado actual del tablero con las piezas y posiciones respectivas, genera una cadena de caracteres llamada *tablero* de la forma *Pieza1Posición1-Pieza2Posición2-…-PiezaNPosiciónN,* con $0 < N < 11$. Esta cadena de caracteres nueva se escribe sobre *partidasguardadas.txt* junto con la fecha en que fue jugada la partida, la última actualización del cronómetro, el nivel y un número que identifica a la partida. Luego, en el menú principal, si el jugador elige cargar una partida, se le mostrará en pantalla los distintos tableros guardados previamente con la información mencionada anteriormente. También, en el nivel fácil el usuario debe resolver el tablero en tres minutos. Para llevar el tiempo, mediante la función *genTimer* se genera un temporizador o reloj en cuenta regresiva en la ventana *INGAME/*menu de juego que le indica al usuario el tiempo disponible para resolver el juego, convirtiéndose el texto que indica el tiempo en rojo cuando este es menor a 30 segundos. Dicho temporizador depende de la variable *TIMER*, la cual depende del nivel escogido por el usuario. La relación entre *TIMER* y el cronómetro es que, gracias a que *TIMER* está en segundos, la división entera *TIMER*/60 devuelve los minutos y la operación *TIMER* mod 60 devuelve los segundos restantes. De esta manera se obtiene el formato minutos:segundos. Cuando el usuario desee pausar el tiempo el tablero se coloca en blanco, esto siendo implementado usando una variable de tipo *bool* llamada *jugar* que al ser verdadero permite al usuario interactuar con el juego y cargar los movimiento. Cuando *jugar* sea falso, inhabilita al jugador de ingresar movimiento y de visualizar el tablero, solo deteniéndose el tiempo.

Para el nivel Difícil, el cronómetro está inicializado en un minuto y medio, se permite pausar el juego, guardar partidas mas no deshacer jugadas. Finalmente para el nivel Muy Difícil, las opciones pausar, guardar partida y deshacer no se permiten y se dispone de un máximo de dos minutos para resolver tres tableros consecutivos. A diferencia de los niveles anteriores donde se gana la partida cuando el contador *cont* llega a uno (es decir, solo permanece una pieza en el tablero), en Muy Difícil se deben resolver tres tableros y que *cont* llegue a uno tres veces, aumentando el valor en uno a la variable *wins* cada vez que se resuelve un tablero y finalmente cuando *wins* llega a tres se gana la partida en el nivel Muy Difícil. Los tres tableros que deben resolverse provienen del archivo *partidasnuevas.txt,* generándose de forma aleatoria y a su vez, cuando se resuelve un tablero, se inicializa la matriz *grid* únicamente con caracteres "Z" en sus casillas para poder procesar el siguiente tablero generado.

Cada movimiento que carga el jugador durante la partida es validada por la función *valid* que recibe como parámetros las posiciones de la pieza que va a capturar y la pieza que será capturada. Se verifica que el movimiento cumpla las reglas de *Solitaire Chess.* En particular, para las piezas Alfil y Torre se tuvieron que crear funciones *valAlfil* y *valTorre* que verificarán que un determinado movimiento no tuviese piezas dentro de su trayectoria, es decir, que la pieza no salte a otras. A su vez, para los movimientos de la Dama, la solución a este problema fue la unión de las soluciones para validar los movimientos del Alfil y de la Torre. Después de verificar si el movimiento es válido, mediante la función *getMovement* se actualiza la matriz *grid* con las nuevas posiciones de las piezas después de procesar el movimiento que cargó el usuario y cada vez que se realizan movimientos válidos, *cont* que es una variable de tipo int va restando su valor inicial el cual era la cantidad de piezas originalmente en el tablero y por cada captura disminuye su valor en una unidad.

Para reconocer que la partida efectivamente ha terminado y así poder declarar al jugador como ganador o perdedor, se tiene una variable booleana *FINISHED* en la función principal del juego el cual es verdadero cuando el temporizador llega a "0:00" (declarándose perdedor el usuario) o cuando el contador de piezas presentes en el tablero *cont* llega a 1 para los niveles Entrenamiento, Fácil y Difícil, siendo esta la condición para declarar un ganador en esos niveles. Para el nivel Muy difícil se gana la partida cuando el contador de tableros resueltos *wins* llega a tres, declarando *FINISHED* en este caso como verdadero.

Finalmente, cada vez que el jugador gana una partida en uno de los cuatro niveles, se registra ese "récord" en un archivo llamado *x.txt* donde x es un número entre 1 y 4 refiriéndose a los niveles. Mediante la función *saveRecord* se guarda el record del usuario en el nivel correspondiente, luego se ordena la lista de records junto con los demás usuarios. Al volver al menú principal, mediante el "botón" *4. MOSTRAR RECORDS* el usuario puede acceder a su registro y ver cuantas partidas ganadas lleva por nivel mediante la función *showRecords* que imprime en pantalla la lista de récords del nivel escogido. Si la lista de records es muy larga, ésta se divide en varias páginas. La variable *pag* indica en qué página se encuentra el usuario y la variable *lastPage* indica el número de la última página.

Estado actual del programa:

El grado de operatividad del juego es efectivamente bueno y alto, no presenta error alguno, el usuario puede navegar en el juego sin dificultad ya que todos los "botones" están indicados con sus teclas correspondientes en el teclado y se le indica al jugador cómo debe ingresar los movimientos por teclado, entre otras opciones.

Conclusiones:

En conclusión, los resultados obtenidos fueron positivos, es decir, el juego funciona de forma correcta y esperada. Los distintos archivos que se generan (partidasguardadas.txt, partidasnuevas.txt, X.txt con 0<X<5) contienen los datos deseados. Todas las opciones habilitadas del juego como Pausar, Deshacer, Terminar, Jugar, Cargar Partida, Generar tablero aleatorio, Mostrar Récords, etc, funcionan correctamente y no generan dificultad alguna para el usuario. Se tiene como resultado final el juego *Solitaire Chess* con todas las debidas funciones y requerimientos.

Se puede decir que es la primera vez creando un juego y dados los buenos resultados la experiencia mientras se implementaba el juego fue satisfactoria ya que se adquirieron conocimientos nuevos y útiles para el resto de la carrera, como trabajar con interfaces gráficas, animaciones, habilitar el teclado para la interacción con el usuario, etc. También mantener el loop principal (y/o la función principal) lo mas posible como bloques de código que llaman a otras funciones que hacen el verdadero trabajo fue una buena manera para organizarse como equipo. Para los casos en los que era más fácil escribir el algoritmo en la función principal, se mantuvo cierto orden gracias a la utilización de variables (window, jugar, etc) donde instrucciones if que utilizaban sus posibles estados conservaron una buena estructura del código. En recomendaciones se podría sugerir que antes de que los profesores manden a hacer el prototipo se debería hacer un prelaboratorio y un laboratorio correspondiente a funciones de pygame.

Entre las dificultades presentadas durante la implementación del juego, se encuentra principalmente programar o habilitar la entrada de texto por teclado del usuario, donde cada vez que éste ingresaba un carácter válido debía ser mostrado en pantalla junto con el resto del string. Este subproblema se pudo solucionar efectivamente como se mencionó anteriormente en el diseño, siendo manejado en la función principal del juego.

Bibliografía:

http://stackoverflow.com/ fecha de consulta 26-11-2016
https://commons.wikimedia.org/wiki/Category:PNG_chess_pieces/Standard_transparent fecha de consulta 26-11-2016
http://www.discoveryplayground.com/computer-programming-for-kids/rgb-colors/ **f**echa de consulta 26-11-2016
Making Games with Python and Pygame por Al Sweigart