Proyecto *El juego de la oca*

Fecha de entrega de la versión 2: 27 de noviembre de 2022, a las 23:50

(Imagen extraída de *https://www.juegodelaoca.com/Reglamento/reglamento.htm*)

El objetivo del proyecto es desarrollar un programa que permita jugar a una versión del popular Juego de la oca. Iremos desarrollando diversas versiones que permitirán poner en práctica los distintos conocimientos de la asignatura.

1. Versión 2: Una oca más versátil

Esta versión del juego permitirá que haya más de 2 jugadores, y en ella el tablero no tiene por qué estar prefijado al tablero *tradicional* usado en la versión 1 sino que las posiciones de las casillas especiales del tablero se cargarán desde un fichero de texto y dichas posiciones no tienen por qué coincidir con las *tradicionales*. Eso sí, el tablero cumplirá con los requisitos para seguir manteniendo las reglas del juego descritas en la versión 1. En particular, tendrá 63 casillas (siendo la 63 una oca), seguirá habiendo los mismos tipos de casillas especiales con los mismos efectos, y se mantendrá el número de casillas especiales (salvo en el caso de las ocas, pudiendo haber más o menos que en el tablero *tradicional*).

El programa comenzará con un número de jugadores prefijado (un valor entre 2 y 4), cargará el tablero desde el fichero cuyo nombre indique el usuario a través de teclado y permitirá jugar una partida con dicho tablero y el número establecido de jugadores. Para facilitar las pruebas y la depuración del programa seguiremos manteniendo los dos modos de juego, el normal y el de depuración, controlados de igual forma que en la versión 1.

1.1. Detalles de implementación

A continuación, vamos a enumerar algunos aspectos que hay que tener en cuenta a la hora de implementar la nueva versión y que suponen cambios con respecto a la oca básica implementada en la versión 1:

- El hecho de que el tablero no sea necesariamente el tradicional conlleva la necesidad de cambiar la forma en la que lo representamos. Ahora ya no usaremos constantes para mantener representadas las posiciones de las casillas especiales porque dichas casillas no están en unas posiciones prefijadas, sino que sus posiciones dependen del tablero que se cargue desde fichero al comienzo de la ejecución del programa. Necesitamos saber qué tipo de casilla tenemos en cada una de las 63 posiciones del tablero y usaremos un tipo array para representar nuestro tablero.
- En la versión 1 solo había dos jugadores y podíamos permitirnos tener sus posiciones guardadas en dos variables independientes. Ahora, el hecho de poder tener más de dos jugadores invita a usar un tipo array para guardar la posición de cada jugador en una componente del array.
- Al tener que cargar el tablero desde un fichero de texto necesitaremos incluir código apropiado para configurarlo a partir del contenido del fichero.
- El hecho de que las posiciones de las casillas especiales no estén prefijadas conlleva cambios en la forma de obtener el efecto que, sobre la posición del jugador, tiene el caer en algunas de las casillas de avance o retroceso. Ahora, por ejemplo, será necesario buscar por el tablero dónde está la siguiente oca ya que no hay un algoritmo que permita computar dónde está la siguiente a aquella en la que ha caído el jugador; igualmente también habrá que buscar la casilla con los primeros dados si se ha caído en los segundos ya que, por defecto, los primeros dados no están en la posición 26 del tablero; etc.
- El hecho de que podamos tener más de 2 jugadores hará necesario cambiar la manera simplista con la que pudimos manejar el efecto de las penalizaciones en la versión 1. En la versión 1 decidimos no representar los turnos de penalización de cada jugador sino que computábamos las tiradas de las que podía disponer el jugador que tenía el turno al principio del mismo (1 por defecto, y, si era el caso, tantas tiradas extra como penalizaciones hubiese recibido el rival en su turno fruto de caer en una casilla con ese efecto). La posibilidad de que existan tres o más jugadores hace inviable que las penalizaciones de uno se conviertan en tiradas extra de los rivales debido a la casuística ampliada que se produce. Ahora cada jugador deberá conocer los turnos de penalización que tiene (0 por defecto) y cada vez que le llegue el dado jugará o no en base a esa información. En cada turno el jugador contará con 1 tirada por defecto, a la que podrán sumarse más tiradas en caso de ir cayendo en casillas con premio. Para guardar el número de turnos de penalización que tiene cada jugador también usaremos un array.
- Por último, será necesario visualizar el tablero para conocer la disposición de las casillas. Además, mejoraremos la interfaz visualizando en él la colocación de los jugadores, que irá cambiando a medida que vayan tirando el dado.

Como consecuencia de todo lo anterior, veamos las indicaciones a seguir para realizar la implementación de esta segunda versión.

Seguiremos manteniendo algunas constantes de la versión 1, como: `NUM_CASILLAS = 63` para representar la última casilla; las que representan el número de turnos de penalización (turnos sin tirar), por ejemplo, `TURNOS_POSADA = 1` representa la penalización por caer en la posada; y la constante `MODO_DEBUG` para establecer el tipo de modo (normal o depuración) en el que se ejecutará el programa. Y, en caso de no haberlo hecho ya en la versión 1, añadiremos al menos la constante `NUM_JUGADORES` de tipo `int` que establece el número de jugadores para la partida.

Definiremos los siguientes tipos:

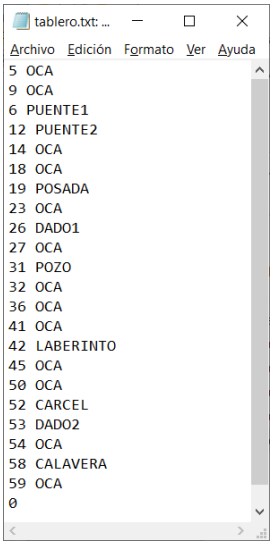
- El tipo enumerado `tCasilla` para representar los diferentes tipos de casillas del tablero:
  ```
  typedef enum {NORMAL, OCA, PUENTE1, PUENTE2, POZO, POSADA, LABERINTO, DADO1, DADO2, CARCEL, CALAVERA} tCasilla;
  ```
  Obsérvese que en el tipo hay dos valores asociados a puentes: `PUENTE1` representa el primer puente, es decir, el puente que se encuentra más cerca del principio del tablero, y `PUENTE2` representa al segundo puente, esto es, al que se encuentra más cerca del final del tablero. Lo mismo ocurre con los dados.
- El tipo array `tTablero` para representar la secuencia de 63 casillas que componen el tablero:
  ```
  typedef tCasilla tTablero[NUM_CASILLAS];
  ```
- El tipo array `tJugadores`
  ```
  typedef int tJugadores[NUM_JUGADORES];
  ```
  cuyas variables pueden ser usadas tanto para representar la posición de cada jugador en el tablero como para representar sus penalizaciones, ya que tanto la posición como la penalización son valores enteros. Por ejemplo, si declaramos la variable `posiciones` de tipo `tJugadores` para representar las posiciones de los jugadores en el tablero, `posiciones[i]` contendrá la posición del tablero en la que se encuentra el jugador `i+1` (recuerda que los arrays se indexan comenzando en 0 y asumimos que los jugadores se identifican con valores que van de 1 a `NUM_JUGADORES`). De manera similar, si declaramos la variable `penalizaciones` de tipo `tJugadores` para representar las penalizaciones de los jugadores, el valor de la componente `penalizaciones[i]` será el nº de turnos de penalización que tiene el jugador `i+1`.

Hay subprogramas de la versión 1 (`tirarDado`, `tirarDadoManual`) que no cambian. Otros habrá que modificarlos ligeramente. Es el caso del subprograma `esMeta`, ya que la meta se encuentra ahora en la posición `NUM_CASILLAS-1` del array que representa al tablero, o del subprograma `quienEmpieza` que habrá que actualizarlo para que devuelva un número en el intervalo [1, `NUM_JUGADORES`] determinado de manera aleatoria, indicando el jugador que comienza la partida. También habrá que actualizar el código encargado del cambio de turno para que permita manejar la alternancia de los turnos entre los `NUM_JUGADORES` jugadores de la partida.

Además, implementaremos, entre otros, los subprogramas que se indican a continuación.

Subprogramas que permiten configurar el tablero

- `void iniciaTablero(tTablero tablero)`: inicializa todas las componentes del tablero al valor `NORMAL` del tipo `tCasilla`, salvo la componente `NUM_CASILLAS-1` que se inicializará al valor `OCA`.
- `bool cargaTablero(tTablero tablero)`: solicita la introducción por teclado del nombre del fichero de texto que contiene la disposición de las casillas especiales (exceptuando la oca de la casilla `NUM_CASILLAS`) y va actualizando las posiciones correspondientes del tablero `tablero`, que se recibe inicializado. Devuelve `true` si la apertura del fichero de texto ha sido correcta y `false` en otro caso.



El fichero estará formado por tantas líneas como casillas especiales haya, de manera que en cada línea figura la posición dentro del tablero físico (valor de 1 a `NUM_CASILLAS-1`) y el identificador de la casilla especial. El fichero termina con el centinela 0 como valor de posición. Si el fichero existe, el contenido es completamente correcto. Es decir, el programa puede asumir que el contenido del fichero es conforme a las especificaciones que se indican en el enunciado y, por lo tanto, el fichero contiene información para generar un tablero válido.
En la figura puede verse el fichero tablero.txt cuyo contenido coincidiría con el tablero *tradicional*.

Subprogramas que sustituyen a otros implementados en la versión 1 relacionados con la identificación de casillas especiales y con el efecto provocado por las mismas

- `bool esCasillaPremio(const tTablero tablero, int casilla)`: devuelve un valor que indica si en la posición `casilla` del tablero hay –`true`– o no –`false`– una casilla de las que provocan avance o retroceso y tienen 1 tirada de premio asociada (es decir, oca, uno de los puentes, o uno de los dados). Este subprograma combina la funcionalidad de los subprogramas `esOca`, `esPuente` y `esDados` de la versión 1, adaptados a la representación usada en la versión 2.
- `void efectoTirada(const tTablero tablero, int& casillaJ, int& penalizacionJ)`: analiza si la casilla situada en la posición `casillaJ` del tablero es una casilla especial y determina su efecto. Si es una casilla especial de retroceso –calavera, laberinto– o de avance o retroceso con premio asociado –oca, uno de los puentes, uno de los dados– actualiza `casillaJ` a la posición del tablero a la que se salta; además, mostrará por pantalla la casilla en la que se está inicialmente y la posición a la que se avanza o se retrocede. Si es una casilla especial de penalización –posada, prisión, pozo– devuelve en `penalizacionJ` los turnos de penalización obtenidos; y además mostrará la casilla en la que se está y las penalizaciones obtenidas. Este subprograma combina la funcionalidad de los subprogramas `efectoTiradas` y `efectoPosicion` de la versión 1, adaptados a la representación usada en la versión 2. En su implementación usará el subprograma `saltaACasilla` descrito a continuación.
- `int saltaACasilla(const tTablero tablero, int casillaActual)`: recibe en `casillaActual` una posición del tablero en la que hay una casilla especial de las que provocan un avance o un retroceso y devuelve la posición del tablero a la que se salta. Como ahora, a diferencia de lo que ocurría en la versión 1, las posiciones de las casillas especiales no están fijas, la localización de la posición a la que se salta puede requerir hacer una búsqueda[1]. Si se trata de una casilla de avance tipo oca, el primer puente, o el primer dado, la posición a la que se salta se localiza buscando desde donde se está hacia

[1] No es necesario buscar por el tablero en el caso del laberinto o de la calavera, ya que en el primer caso se retroceden 12 posiciones y en el segundo se va directo a la casilla de inicio. En los demás casos, sí es necesario.

el final del tablero; si es una casilla de retroceso como el segundo puente o el segundo dado, la búsqueda debe dirigirse desde donde se está hacia el principio del tablero.
Es recomendable implementar subprogramas que generalicen esas búsquedas, parametrizándolos con la casilla que se quiere buscar. De esta forma pueden reutilizarse a la hora de implementar el subprograma `saltaACasilla`. Por lo tanto, se recomienda implementar un subprograma que generalice la búsqueda desde una posición del tablero avanzando –es decir, hacia el final del tablero–, parametrizándolo con la casilla a buscar; y otro que generalice la búsqueda desde una posición del tablero retrocediendo –es decir, hacia el principio del tablero–, parametrizándolo con la casilla a buscar. Dichos subprogramas son, respectivamente, los siguientes:

- `void buscaCasillaAvanzando(const tTablero tablero, tCasilla tipo, int& posicion)`: se encarga de buscar la casilla `tipo` desde la posición `posicion` del tablero hacia el final del mismo. Actualiza `posicion` a la posición localizada.
- `void buscaCasillaRetrocediendo(const tTablero tablero, tCasilla tipo, int& posicion)`: se encarga de buscar la casilla `tipo` desde la posición `posicion` del tablero hacia el principio del mismo. Actualiza `posicion` a la posición localizada.

Subprogramas para la lógica de las tiradas y de la partida

- `void iniciaJugadores(tJugadores casillasJ, tJugadores penalizacionesJ)`: inicializa los arrays `casillasJ` y `penalizacionesJ` que almacenan, respectivamente, la posición en el tablero y el número de turnos sin jugar (penalización) de cada jugador. Inicialmente todos los jugadores estarán en la casilla de partida y no tienen penalización alguna.
- `void tirada(const tTablero tablero, int& casillaActual, int& penalizacion)`: recibe el tablero y la posición del jugador que va a tirar el dado (`casillaActual`) e implementa la lógica de la tirada. Una vez mostrada la posición actual, lanza el dado (aleatoriamente si se está en modo normal de juego o manualmente si se está en modo depuración) y muestra la posición a la que avanza el jugador. Si no alcanza la meta se determina el efecto que la casilla a la que se ha avanzado puede provocar, utilizando para ello el subprograma `efectoTirada` descrito anteriormente. `casillaActual` quedará finalmente actualizada a la posición alcanzada por el jugador tras todo este proceso y en `penalización` se devolverá la penalización obtenida por el jugador al valorar el efecto del avance (si alguna).
- `int partida(const tTablero tablero)`: recibe el tablero de juego e implementa toda la partida. Para ello:
  - pondrá a los jugadores en disposición de empezar el juego (usando el subprograma `iniciaJugadores`),
  - decidirá qué jugador empieza (usando el subprograma `quienEmpieza`),
  - y mientras que no se alcance la meta realizará una tirada el jugador que tiene el turno (usando el subprograma `tirada`) siempre que no esté penalizado (si lo estuviese se reduciría su penalización en un turno) y se avanzará el turno al siguiente jugador (si procede).

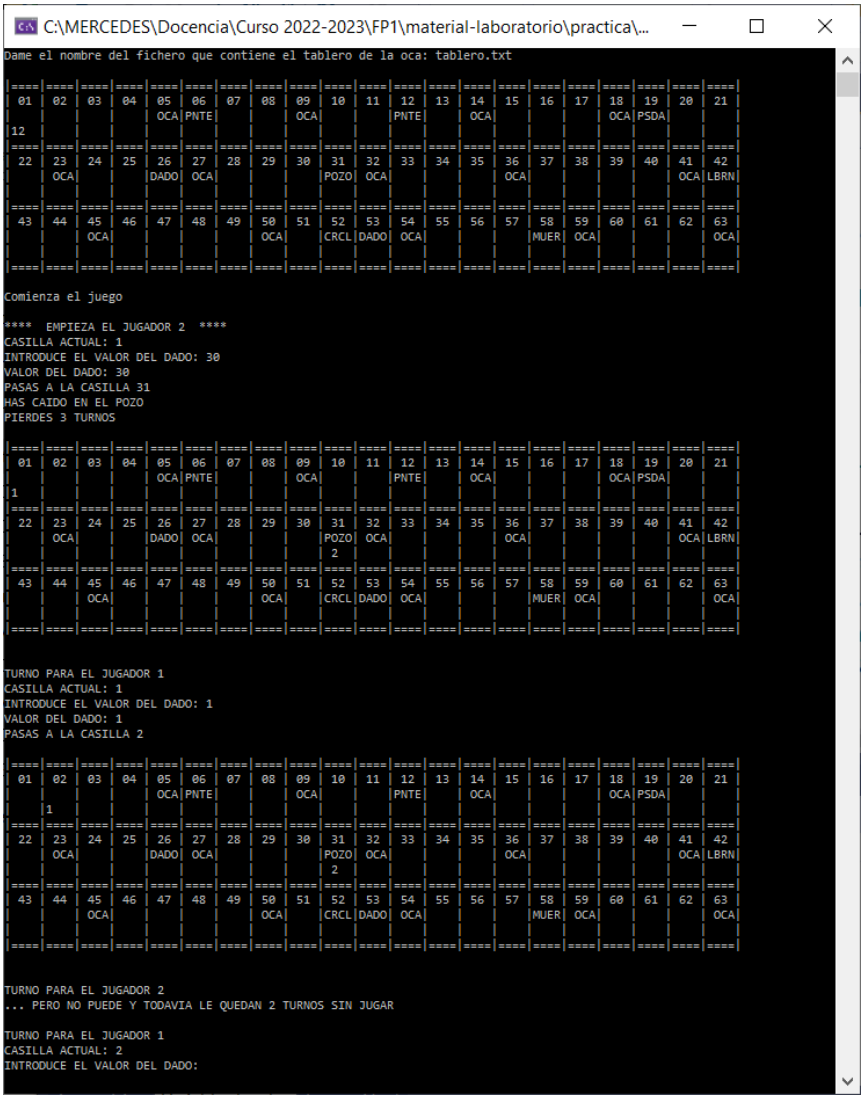
El tablero se irá mostrando convenientemente para ir reflejando cada cambio que se produzca. Y al finalizar el subprograma devolverá un valor que indique cuál de los jugadores ha ganado la partida (aquel que haya realizado la última tirada).

Por último, se necesita un subprograma encargado de la visualización del tablero y de la disposición de los jugadores en el mismo: `void pintaTablero(const tTablero tablero, const tJugadores casillasJ)`. Este subprograma y otros auxiliares utilizados en su implementación, se proporcionan ya implementados en la plantilla que acompaña a este enunciado.

En las páginas siguientes pueden verse algunos fragmentos de ejemplos de ejecución.

1.2. Ejemplos de ejecución

En este primer ejemplo (2 jugadores, tablero *tradicional*, modo depuración) puedes ver el comportamiento del programa cuando el que inicia la partida cae en una casilla de penalización (el pozo).



Empieza el jugador 2: se introduce un 30 como tirada y ello provoca que el jugador 2 pase a la casilla 31.

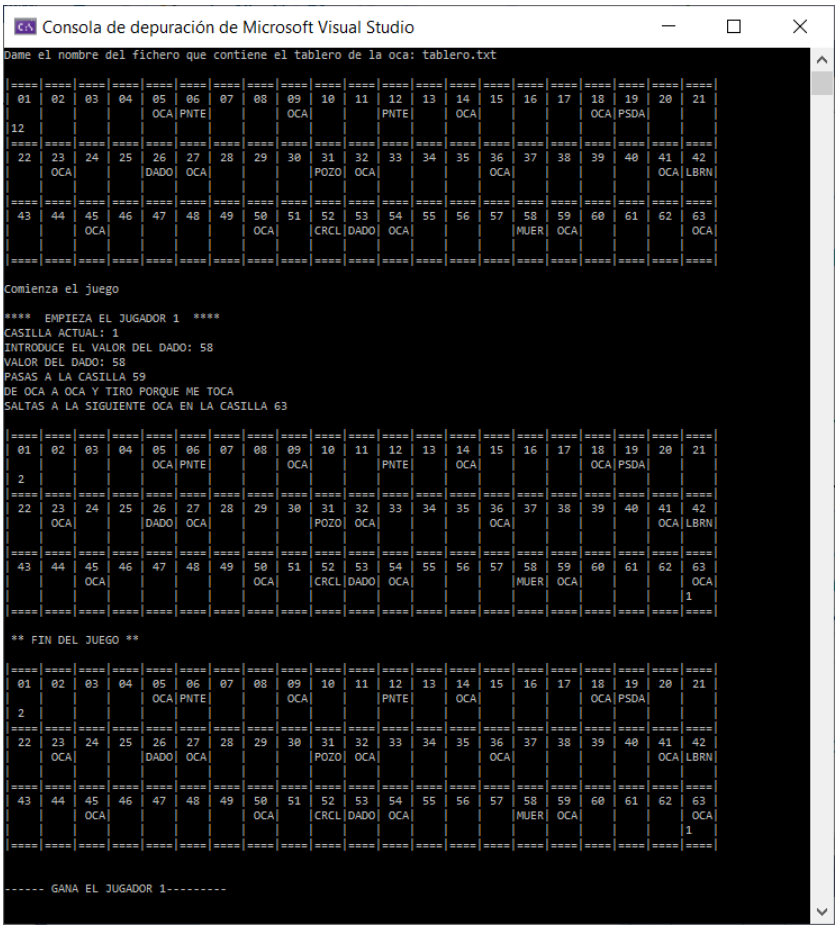
Esta casilla corresponde al POZO, por lo que el jugador 2 pierde 3 turnos.

Comienza a jugar el jugador 1: se introduce el valor 1. El jugador 1 pasa a la casilla 2.

Le llega el turno al jugador 2, que está penalizado y no puede tirar. Le quedan por delante 2 turnos más sin tirar.

Pasa el turno al jugador 1, que podrá introducir el valor del dado.

En este segundo ejemplo (2 jugadores, tablero *tradicional*, modo depuración) puedes ver lo que ocurre cuando el que inicia el juego cae en casilla con premio (la penúltima oca).



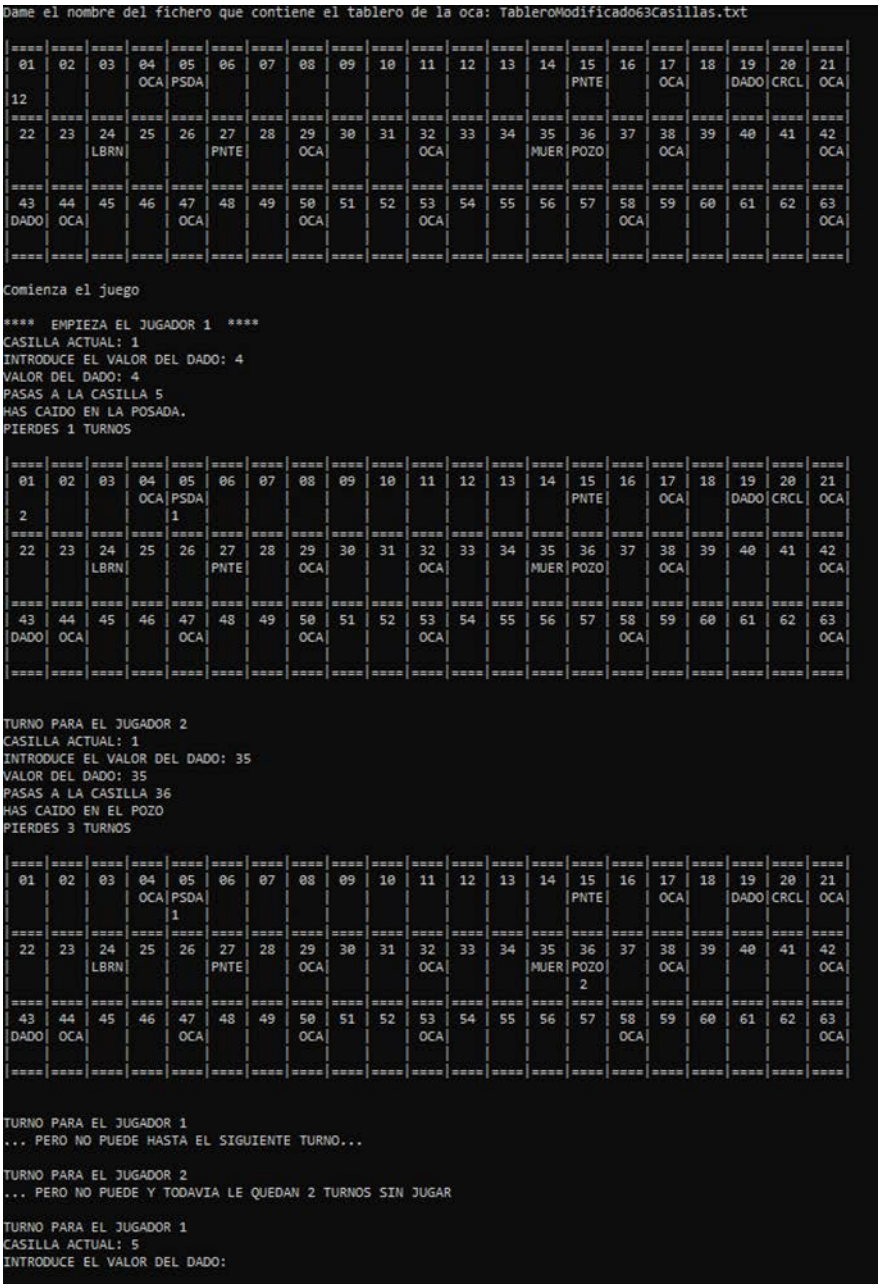
Empieza el jugador 1: se introduce un 58 como tirada y ello provoca que el jugador 1 pase a la casilla 59.

Esta casilla corresponde a una OCA, por lo que el jugador 1 salta a la siguiente OCA que es la de la casilla `NUM_CASILLAS`.

Acaba el juego sin que el jugador 2 haya podido participar.

Gana el jugador 1.

En este tercer ejemplo (2 jugadores, tablero *no tradicional*, modo depuración) se observa el efecto de caer ambos jugadores en casillas que conllevan penalización (posada, pozo) y cómo afecta a los turnos de uno y otro jugador.



Empieza el jugador 1: se introduce un 4 como tirada y ello provoca que el jugador 1 pase a la casilla 5.

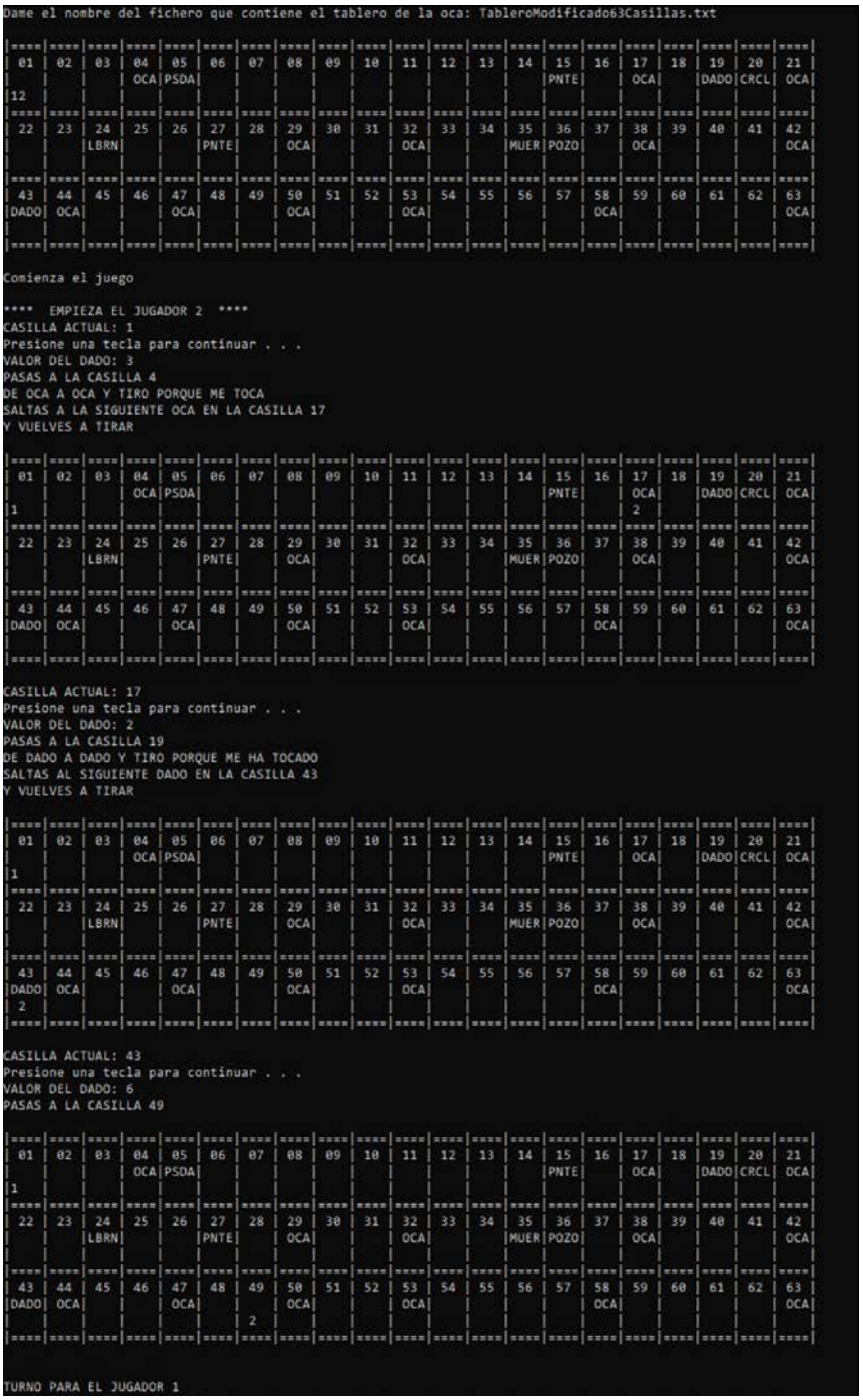
Esta casilla corresponde a la POSADA, por lo que el jugador 1 pierde 1 turno.

Comienza a jugar el jugador 2: se introduce el valor 35. El jugador 2 pasa a la casilla 36 que es el POZO. Esto provoca la pérdida de 3 turnos.

Como se ve en el tablero, ambos jugadores están en casillas penalizadas y ninguno puede tirar porque aún no se han terminado sus turnos de penalización.

El jugador 1 ya puede volver al juego, pues ya terminó su penalización.

Este último ejemplo (2 jugadores, tablero *no tradicional*, modo normal) permite observar diversas tiradas de un mismo jugador dentro de su turno.



Empieza el jugador 2: el dado saca un 3, y pasa a la casilla 4, que es una OCA.

De OCA (casilla 4) a OCA (casilla 17) y el jugador 2 vuelve a tirar.

Esta vez el dado vale 2 y el jugador 2 pasa a la casilla 19 que es la casilla especial de DADO y cuyo efecto es un salto de la casilla 19 a la 43 y disponer de una tirada más.

Así que el jugador 2 vuelve a tirar los dados. Esta vez el valor es 6 y le permite avanzar hasta la casilla 49.

Y le llega el turno al jugador 1 ...