Final report:

L'objectiu de les pràctiques de l'assignatura d'Estructures de Dades i Algorismes era elaborar un programa per al popular joc de "Snakes and Ladders". Dita pràctica, es dividia en tres laboratoris i, en cada un d'ells s'havien d'implementar certes funcions i anar escrivint i millorant el codi.

El joc consisteix en un tauler bidimensional de *X* caselles amb una casella inicial i una casella final. El jugador per avançar de casella ha de tirar un dau de sis cares i avançar tantes caselles com indica el dau. No obstant, hi ha certes caselles que perjudiquen o ajuden al jugador. És el cas de les *Snakes* (serps) i les *Ladders* (escales). Si el jugador cau a una casella de *Snake*, haurà de retrocedir fins a la casella que indica la *Snake*. Per el contrari, si cau a una casella de *Ladder*, haurà d'avançar fins a la casella que indica la *Ladder*.

Una vegada ja sabem en que haurà de consistir el codi del joc, explicarem de forma detallada el que hem fet a cada un dels laboratoris i el funcionament del codi.

El primer laboratori fou el més senzill de tots, ja que feia la funció d'introducció al projecte. L'objectiu del laboratori era crear les estructures de dades per a les caselles, el taulell de joc i l'estat del joc (*tile*, *board* i *state*). La primera estructura de dades que vam crear fou la de caselles (*tile*). Aquesta estructura de dades està formada per tan sols dos paràmetres: un enter *position* i un altre enter *target position*. El paràmetre *position* indica la posició de la casella (ens diu si és la primera casella, la tercera, la sisena, etc) i el paràmetre *target position* ens indica, en el cas de que la casella sigui una *Ladder* o una *Snake*, la casella a la que s'enviarà al jugador si cau a la casella. A continuació, vam crear l'estructura de dades del taulell (*board*). Aquesta estructura de dades està formada per: un enter que indica el nombre de columnes, un altre enter que indica el nombre de files i un array doble (una matriu) de caselles (*tiles*) que emmagatzema la informació de totes les caselles del taulell. Finalment, vam crear l'estructura de dades de l'estat del joc (*state*). Aquesta estructura de dades està formada per: un enter que indica la posició del jugador, un enter que indica si el joc ha acabat o no i un paràmetre taulell (*board*) definit a l'estructura de dades anterior.

L'objectiu del segon laboratori era implementar totes les funcions relacionades amb les caselles (*tiles*), el taulell de joc (*board*) i l'estat del joc (*state*). En relació a les caselles, vam haver d'implementar set funcions:

- funció init_tile (Tile * tile, int position) → aquesta funció inicialitza una casella a partir de dos paràmetres: una estructura de dades buida de *Tile* i un enter per a la posició

de la casella. Aquesta funció tan sols assigna valors a les variables de l'estructura de dades de *Tile*.

- funció get_position (Tile* tile) → aquesta funció tan sols retorna un enter, el valor de la posició de la *Tile* que s'introdueix com a paràmetre.

- funció get_target_position (Tile * tile) → igual que la funció anterior, aquesta funció tan sols retorna un enter, el valor de la posició objectiu (*target position*) de la *Tile* que s'introdueix com a paràmetre.

- funció set_target_position (Tile * tile, int target_position) → aquesta funció serveix per declarar la target position de les caselles (*Tiles*) que són *Ladders* o *Snakes*. La funció canvia el valor *target_position* de l'estructura de dades de *Tile* que hem introduït com a paràmetre pel nou valor enter *target_position*.

- funció clear_target_positon (Tile * tile) → aquesta funció elimina el valor de *target position* de la casella (*Tile*) introduïda com a paràmetre.

- funció is_ladder (Tile * tile) → aquesta funció retorna un valor booleà en funció de si la casella introduïda com a paràmetre és una *Ladder* (la *target position* és major que la *position* de la casella).

- funció is_snake (Tile * tile) → aquesta funció retorna un valor booleà en funció de si la casella introduïda com a paràmetre és una *Snake* (la *target position* és menor que la *position* de la casella).

En relació al taulell de joc, vam implementar les següent funcions:

- funció init_board (Board * board, int rows, int columns) → si observem el codi, veurem que apareix dues vegades. Aquest fet es degut a que, per a l'entrega del segon laboratori, el vàrem escriure usant tres bucles for. Malgrat, durant el tercer laboratori, vàrem detectar que el codi estava equivocat i el vàrem re-escriure. El codi que no està comentat, és el codi que sense error. Aquesta funció serveix per inicialitzar totes i cada una de les caselles del taulell. El bucle for es repeteix tantes vegades com caselles hi ha al taulell (files * columnes) i, a cada iteració del bucle, s'inicialitza una nova casella.

- funció get_rows (Board * board) → aquesta funció retorna el valor enter de *rows* (files) del paràmetre board (l'estructura de dades definida al primer laboratori).

- funció set_rows (Board * board, int rows) → aquesta funció estableix el valor de la variable *row* de l'estructura de dades board introduïda com a paràmetre.

- funció get_columns (Board * board) → aquesta funció retorna el valor enter de *columns* (columnes) del paràmetre board (l'estructura de dades definida al primer laboratori).

- funció set_columns (Board * board, int rows) → aquesta funció estableix el valor de la variable *columns* de l'estructura de dades board introduïda com a paràmetre.

- funció get_size (Board * board) → aquest funció retorna el resultat de la multiplicació entre el nombre de columnes i el nombre de files. És a dir, el nombre de caselles del taulell de joc.

- funció get_tile_at (Board * board, int position) → aquesta funció serveix per trobar el punter d'una casella (*Tile*) del taulell prèviament inicialitzada.

En relació a l'estat del joc, vam implementar les següents funcions:

- funció init_state (State * state, Board * board) → aquesta funció serveix per inicialitzar el joc. La funció defineix la variable booleana que indica si el joc ha acabat com a falsa i la posició del jugador com a zero (a la casella inicial).

- funció set_current_position (State * state, int position) → aquesta funció estableix la variable user_position de l'estructura de dades de *State* al valor enter *position* introduït com a paràmetre.

- funció get_current_positon (State * state) → aquesta funció retorna el valor enter de la variable user_position de l'estructura de dades *State* introduïda com a paràmetre.

- funció set_finished (State * state, int finished) → aquesta funció estableix la variable game_finished de l'estructura de dades de *State* al valor enter *finished* introduït com a paràmetre.

- funció is_finished (State * state) → aquesta funció retorna el valor de la variable game_finished de l'estructura de dades *State* introduïda com a paràmetre.

- funció move (State * state, int dice_value) → si observem el codi, veurem que apareix dues vegades. Aquest fet es degut a que, per a l'entrega del segon laboratori, el vàrem escriure de tal forma que en certes circumstàncies el codi retornava error. Malgrat, durant el tercer laboratori, vàrem detectar que el codi estava equivocat i el vàrem re-escriure. El codi que no està comentat, és el codi que sense

error. Aquesta funció primer comprova si el joc ha acabat per sortir de la funció, si el joc no ha acabat, canvia la posició del jugador per la posició resultant de sumar la posició actual i el resultat de la tirada del dau. Posteriorment, la funció comprova si la posició nova del jugador es tracta d'una *Snake* o una *Ladder* i, en cas de ser-ho, mou al jugador a la *target_position* indicada per la casella.

La segona part del segon laboratori consistia en implementar tres funcions que permeten inicialitzar un taulell a partir d'un fitxer. Les funcions que vam implementar foren:

- funció load_board (Board * board, char * path) → aquesta funció llegeix el fitxer indicat per la variable path i emmagatzema la informació del fitxer a l'estructura de dades *board* definida al primer laboratori.

- funció read_tile_line (Board * board, File * fd) → aquesta funció llegeix el fitxer introduït com a paràmetre i emmagatzema la informació de les caselles *Ladder* i *Snake* a l'estructura de dades *Tile* corresponent.

- funció read_board_line (Board * board, File * fd) → aquesta funció, igual que la funció anterior, llegeix el fitxer introduït com a paràmetre i emmagatzema la informació que indica el taulell a inicialitzar a l'estructura de dades *board*.

- funció check_tile_data (int position, char type, int target, int board_size) → aquesta funció serveix per comprovar que la informació del fitxer a llegir per inicialitzar el taulell és correcta. És a dir, si la posició i la posició objectiu de les caselles està dins la mida del taulell, si s'ha usat la lletra pertinent per indicar si una casella és una *Ladder* o una *Snake*, etc.

- funció init_basic_board (Board * board) → aquesta funció serveix per inicialitzar el taulell de joc bàsic de nou caselles (3x3) mitjançant un bucle for per iterar a través de totes les posicions possibles de les caselles.

Finalment, el tercer laboratori consistia en definir dues estructures de dades i implementar certes funcions. Primer vam implementar les estructures de dades de *step* i de *sequence*. L'estructura de dades *step* està formada per: un valor enter que ve determinant per la tirada de dau, un altre valor enter que determina la posició resultant una vegada s'ha sumat a la posició actual el resultat de la tirada del dau i una estructura de dades *step* per a la següent jugada. L'estructura de dades *sequence* està formada per: una estructura de dades *step* de la primera jugada de la partida, una estructura de dades *step* de la darrera jugada de la partida i un enter *size* que ens indica el nombre de jugades que s'han realitzat a la partida.

Les funcions que vam implementar en relació a la seqüència de joc són les següents:

- funció init_sequence (Sequence * sequence) → aquesta funció inicialitzar l'estructura de dades *Sequence* per poder emmagatzemar els valors pertinents a les variables posteriorment.

- funció add_step_as_first (Sequence * sequence, int position, int dice_values) → aquesta funció serveix per afegir la primera *Step* a la *Sequence* mitjançant una memòria dinàmica (ja que no es pot saber la mida final de la *Sequence* fins una vegada ha acabat el joc).

- funció add_step_as_last (Sequence * sequence, int position, int dice_values) → aquesta funció serveix per afegir el darrer *Step* a la *Sequence* mitjançant una memòria dinàmica (ja que no es pot saber la mida final de la *Sequence* fins una vegada ha acabat el joc).

- funció get_sequence_size (Sequence * sequence) → aquesta funció retorna la mida de la *Sequence* accedint a l'estructura de dades introduïda com a paràmetre.

- funció clear_sequence (Sequence * sequence) → aquesta funció serveix per esborrar els valors de les variables de l'estructura de dades *Sequence* introduïda com a paràmetre.

- funció print_sequence (Sequence * sequence) → aquesta funció serveix per imprimir a la consola tots els *Steps* de la *Sequence* mitjançant un bucle while que itera fins que es troba un *Step* buit (NULL).

A continuació, parlarem sobre les possibles extensions que es poden implementar al codi. En primer lloc, es pot millorar el programa afegint més jugadors. D'aquesta forma, el joc es convertiria en un joc per torns en el qual cada jugador hauria de tenir assignada una estructura de dades *State* per garantir que el joc continua mentre cap dels dos jugadors ha guanyat, és a dir, mentre la variable *game_finished* de cap dels jugadors és certa. Per altra banda, es pot millorar el joc fent que el taulell generat no tingui la forma d'un taulell quadrat bidimensional. Per exemple, es podria fer que el taulell es generi: en forma d'espiral, de forma inversa (la casella de sortida es troba a la part superior del taulell i la casella final es troba a la part inferior), etc. Però degut a falta de temps, no hem implementat noves funcions al nostre programa.

Finalment, parlarem sobre les impressions generals que hem tingut de la pràctica. La pràctica ens ha servit per entendre millor com elaborar un codi complex que, gràcies a

funcions més senzilles definides prèviament, es converteix en un codi més senzill i millor optimitzat. Considerem que es tracta d'un treball final prou complexe com per entendre la gran quantitat de possibilitats que estan al nostre abast a l'hora de programar. Dividir l'entrega en tres laboratoris diferents ens ha permès poder avançar de forma contínua i moderada. Malgrat, considerem que hagués estat pertinent rebre la qualificació del segon laboratori abans de la data d'entrega del tercer laboratori, ja que d'aquesta manera podríem haver comprovat que el codi escrit al segon laboratori estava ben fet. Nosaltres ens hem adonat de que certes part de codi que vam entregar al segon laboratori no eren del tot correctes i les hem pogut corregir, però hagués estat molt més senzill i adient haver rebut l'ajuda del professor per a les correccions.