**Andrei Petruk**

**Procedural Programming Laboratory Project**

**The Snakes and Ladders**

1. At the start of the game each player is located just off the board.
2. In turn each player rolls a dice and moves the number of space dictated by the dice. For example:
   * **If at the start of the game the player rolls five then they move to square number 5.**
   * **If the player is located at square 8 and rolls a 4 then they move to square number 12.**
3. If a player lands at the bottom of a ladder he climbs up the ladder and thus does not have to cover the whole path.
4. If a player land on a Snake's mouth he slips down to the snake’s tail.
5. Once a player has completed his/her move the status of the game should be displayed to screen. Then the next player repeats steps 2-4.
6. This process continues until one player reaches square number 100 – At which point they are declared the winner.

**Steps:**

**1**

* 100 squares
* Make the board random
* 7 snakes
* 7 ladders
* 2-6 players no of players set by user
* Save status of player
* Dice (random number generator)

**2**

* Give option ( Start new game or load an old game)

**3**

**New Game**

* How many players
* Configure board

Conditions for a ladders and snakes

1. The head must be above the tail

2. Must not be in last position

Initialise methods to:

* initialize board
* print out the positions of all snakes and ladders in the game
* generate a random game board
* generate random number from 1 - 6
* allow the user to start a new game without restarting the program
* move player along the board
* allow the user to save to one of four files
* allows the user to load a previously saved game
* gets the position a player, snake, ladder

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<time.h>

#pragma once // for using Booleans in vs 2010

#define false 0

#define true 1

#define bool int

typedef struct player\_t {

int row;

int col;

bool isTurn; // saving game

} player\_t;

typedef struct snake\_t {

int headRow;

int headCol;

int tailRow;

int tailCol;

} snake\_t;

typedef struct ladder\_t {

int headRow;

int headCol;

int tailRow;

int tailCol;

} ladder\_t;

void boardSetUp();

void PositionOfSnakesLadders();

void boardGenerator();

int rollDice();

void newGame();

void move();

void save();

bool load();

int getPos(int, int); // gets curent position

char\*\* gameBoard; // pointer

player\_t\* players;

snake\_t\* snakes;

ladder\_t\* ladders;

FILE\* file; //file pointer for saving and loading the game

int boardRows = 10;

int boardCols = 10;

int activePlayer; // the current player that is active

int numSnakesAndLadders;

int numPlayers;

bool gameOver = false; // checks if the game is over

void main() {

bool loaded = false;

int i;

char choice;

char game; // choice to start a new game

do {

printf("Press (L) To load a previously saved game.\n");

printf("Press (N) To start new game.");

scanf(" %c", &choice); // prompt to load game

} while (choice != 'n'

&& choice != 'N'

&& choice != 'l'

&& choice != 'L');

switch (choice) {

case 'L':

case 'l':loaded = load();

break;

case 'n':

case 'N':printf("Starting a new game!\n");

break;

}

if (!loaded) {

newGame();

} // no game was loaded, start a new game

activePlayer = 0; //being the active player

while (choice != 'x'

&& choice != 'X'

&& gameOver == false) {

for (i = 0; i < numPlayers; i++) {

if (players[i].isTurn == true) {

activePlayer = i; // remember the position of the active player for load feature

}

}

printf("\nIt's player %d's turn. \n", (activePlayer + 1));

printf("Select your option: \n");

printf("Press (M) To Trow Dice for Move\n");

printf("Press (P) To Get position of Snakes and Ladders\n");

printf("Press (N) To Start new game\n");

printf("Press (S) To Save game\n");

printf("Press (L) To Load game\n");

printf("Press (X) To Exit without saving.\n");

scanf(" %c", &choice);

players[activePlayer].isTurn = true; // save which player's turn for resume

switch (choice) {

case 's':

case 'S':save(); // save the game

break;

case 'l':

case 'L':load(); // load the game

break;

case 'n':

case 'N':

printf("Are you sure you want to start a new game? (Y/N): ");

// only want to prompt if they make a new game after starting a new game for the first time

scanf(" %c", &game);

if (game == 'y' || game == 'Y') {

newGame(); // start a new game

} else {

printf("Resuming game.");

}

break;

case 'm':

case 'M': move(); //player turn

break;

case 'p':

case 'P':PositionOfSnakesLadders();

break;

case 'x':

case 'X': // game ends here

break;

} // end of switch

} // end of main while loop

getch();

}

void boardSetUp() {

int i, j;

for (i = 0; i < boardRows; i++) {

for (j = 0; j < boardCols ; j++) {

gameBoard[i][j] = ' '; // empty every position on the board

}

}

}

void PositionOfSnakesLadders() {

int i;

int headPos, tailPos;

for (i = 0; i < numSnakesAndLadders; i++) {

headPos = getPos(ladders[i].headRow, ladders[i].headCol);

tailPos = getPos(ladders[i].tailRow, ladders[i].tailCol);

printf("Ladder #%d: Head at pos: %d, bottom at pos: %d\n", (i + 1), headPos + 1, tailPos + 1);

}// show the location of all the ladders on the board

printf("\n");

for (i = 0; i < numSnakesAndLadders; i++) {

headPos = getPos(snakes[i].headRow, snakes[i].headCol);

tailPos = getPos(snakes[i].tailRow, snakes[i].tailCol);

printf("Snake #%d: Head at pos: %d, tail at pos: %d\n", (i + 1), headPos + 1, tailPos + 1);

} // show the location of all snakes on the board

printf("\n");

for (i = 0; i < numPlayers; i++) {

if (getPos(players[i].row, players[i].col) == 0) {

printf("Player #%d hasn't started yet!\n", (i + 1));

} else {

printf("Player #%d at pos: %d\n", (i + 1), getPos(players[i].row, players[i].col) + 1);

}

}

printf("\n");

}

void boardGenerator() {

int i; // counter

int hrow, hcol, trow, tcol; // head row , col and tail row, col

numSnakesAndLadders = 7;

srand(time(NULL));

// create an array of snakes and ladders using dynamic memory allocation

snakes = (snake\_t\*)malloc(numSnakesAndLadders\*sizeof(snake\_t));

ladders = (ladder\_t\*)malloc(numSnakesAndLadders\*sizeof(ladder\_t));

for (i = 0; i < numSnakesAndLadders; i++) {

do {

hrow = rand() % boardRows;

hcol = rand() % boardCols; // randomly assign row and col for head

trow = rand() % boardRows;

tcol = rand() % boardCols; // randomly assign row and col for tail

} while ((hrow < trow) // cannot be in a lower row

|| (hrow == trow) // cannot be on the same row

|| (gameBoard[hrow][hcol] != ' ') // not an empty space

|| (gameBoard[trow][tcol] != ' ') // not an empty space

|| (hrow == (boardRows - 1) && hcol == (boardCols - 1)// must not be the final square

|| (hrow == 0 && hrow == 0)));

ladders[i].headCol = hcol;

ladders[i].headRow = hrow;

ladders[i].tailCol = tcol;

ladders[i].tailRow = trow;

// fill in the details for that ladder

gameBoard[ladders[i].headRow][ladders[i].headCol] = 'L';

gameBoard[ladders[i].tailRow][ladders[i].tailCol] = 'l';

// fill up the gameboard with the positions of the ladders

do {

hrow = rand() % boardRows;

hcol = rand() % boardCols;

trow = rand() % boardRows;

tcol = rand() % boardCols;

} while ((hrow < trow) // must not be in a higher row

|| (hrow == trow) // must not be on the same row

|| gameBoard[hrow][hcol] != ' ' // must be an empty space

|| gameBoard[trow][tcol] != ' ' // must be an empty space

|| (hrow == (boardRows - 1) && hcol == (boardCols - 1) // must not be on the last square

|| (hrow == (0) && hcol == 0)));

snakes[i].headCol = hcol;

snakes[i].headRow = hrow;

snakes[i].tailCol = tcol;

snakes[i].tailRow = trow;

gameBoard[snakes[i].headRow][snakes[i].headCol] = 'S';

gameBoard[snakes[i].tailRow][snakes[i].tailCol] = 's';

// fill up the gameboard with the positions of the snakes

} // for each snake and ladder

} // end of randomizeBoard

int rollDice() {

int rnd;

srand(time(NULL)); // seed

rnd = rand() % 6 + 1; // random number generator

return (rnd);

}

void newGame() {

int i;

do { // how many players

printf("Please enter number of players? (betwen 2-6): ");

scanf("%d", &numPlayers);

} while (numPlayers > 6 || numPlayers < 2);

players = (player\_t\*)malloc(numPlayers\*sizeof(player\_t)); // dynamically create storage for number of players

gameBoard = (char\*\*)malloc(boardRows\*sizeof(char\*));// Dynamically allocate board space

for (i = 0; i < boardRows; i++) {

gameBoard[i] = (char\*)malloc(boardCols \* sizeof(char));

}

boardSetUp(); // initialize the game board

// create the players

for (i = 0; i < numPlayers; i++) {

players[i].col = 0;

players[i].row = 0;

players[i].isTurn = false;

} // initialize all the players in the game

players[0].isTurn = true; // ensure player one is always the first player

printf("\n\n");

boardGenerator(); // creates the game

} // startNewGame

void move() {

char choice;

int i;

int fromPos, toPos;

int rolled;

players[activePlayer].isTurn = false; // only one active player

rolled = rollDice(); // random value from 1-6

printf("\nPlayer %d Rolled a %d!\n", (activePlayer + 1), rolled);

fromPos = getPos(players[activePlayer].row, players[activePlayer].col); // get the position before move

toPos = getPos(players[activePlayer].row, players[activePlayer].col) + rolled; // get the position now at

players[activePlayer].row = toPos / boardCols; // gets the row the player is on

players[activePlayer].col = toPos % boardCols; // gets the col the player is on

if (toPos >= 100) { // player has won

players[activePlayer].row = boardRows - 1; //no index out of bounds exception if player won

players[activePlayer].col = boardCols - 1;

toPos = 99; //

}

printf("Player %d moved from position %d to position %d\n", (activePlayer + 1), fromPos + 1, toPos + 1);

if (gameBoard[players[activePlayer].row][players[activePlayer].col] == 'l') {

// bottom of ladder

for (i = 0; i < numSnakesAndLadders; i++) { // search through ladders

if (getPos(ladders[i].tailRow, ladders[i].tailCol) == toPos) {

players[activePlayer].row = ladders[i].headRow;

players[activePlayer].col = ladders[i].headCol;

printf("Player %d get on ladder and moved to position %d\n",

(activePlayer + 1), getPos(players[activePlayer].row, players[activePlayer].col) + 1);

break; // don't need to look anymore

} // look through ladders

} // for

} // if ladder

else if (gameBoard[players[activePlayer].row][players[activePlayer].col] == 'S') {

// head of snake

for (i = 0; i < numSnakesAndLadders; i++) {

if (getPos(snakes[i].headRow, snakes[i].headCol) == toPos) { // if player landed on head of a snake

players[activePlayer].row = snakes[i].tailRow;

players[activePlayer].col = snakes[i].tailCol;

printf("Player %d get on snake and moved to position %d\n",

(activePlayer + 1), getPos(players[activePlayer].row, players[activePlayer].col) + 1);

break; // no need to search further

} // if landed on snake head

} // search through snakes

} else {

if (getPos(players[activePlayer].row, players[activePlayer].col) == boardRows \* boardCols - 1) {

// will be true if player has passed the last space

printf("Player %d is winer!\n", (activePlayer + 1));

do {

printf("Would you like to play again? (Y/N): ");

scanf(" %c", &choice);

} while (choice != 'n'

&& choice != 'N'

&& choice != 'y'

&& choice != 'Y');

switch (choice) {

case 'n':

case 'N':

printf("The end!\n");

gameOver = true; // end game

break;

case 'y':

case 'Y':newGame();

break;

default:

break;

}

}

}

// other wise assign the next player

if (activePlayer == (numPlayers - 1)) { // if it's the last player

activePlayer = 0; // back to player 1

} else {

activePlayer++; // next player's turn

}

} // roll

int getPos(int row, int col) {

return (row \* boardCols) + col;

}

void save() {

int save, i, j;

printf("Save over which file - 1, 2, 3 or 4 ?: ");

scanf("%d", &save);

switch (save) {

case 1:file = fopen("file1.text", "w");

break;

case 2:file = fopen("file2.text", "w");

break;

case 3:file = fopen("file.text", "w");

break;

case 4:file = fopen("file4.text", "w");

break;

default:

printf("Sorry, please enter either 1,2,3 or 4 for saving the game!\n");

file = NULL;

} // open up one of three files, or give error message

if (file != NULL) { // save over the specified file

fprintf(file, "%d %d %d %d\n", numPlayers, numSnakesAndLadders, boardRows, boardCols);

// save all

for (i = 0; i < numSnakesAndLadders; i++) {

fprintf(file, "%d %d %d %d\n", snakes[i].headCol, snakes[i].headRow, snakes[i].tailCol, snakes[i].tailRow);

} // snake info

for (i = 0; i < numSnakesAndLadders; i++) {

fprintf(file, "%d %d %d %d\n", ladders[i].headCol, ladders[i].headRow, ladders[i].tailCol, ladders[i].tailRow);

}// ladder info

for (i = 0; i < numPlayers; i++) {

fprintf(file, "%d %d %d\n", players[i].col, players[i].isTurn, players[i].row);

} // print all the player information

fclose(file); // close the file

printf("File Saved!\n\n");

} else {

printf("Error saving to the file!\n\n");

}

} // save

bool load() { // load a previously saved game

int choice, i;

if (gameBoard != NULL) {

free(gameBoard);

} // free last allocated gameboard

printf("Which file to load? - 1,2,3 or 4 ?: ");

scanf("%d", &choice);

switch (choice) {

case 1: file = fopen("file1.text", "r");

break;

case 2: file = fopen("file2.text", "r");

break;

case 3: file = fopen("file3.text", "r");

break;

case 4: file = fopen("file4.text", "r");

break;

default:

file = NULL;

printf("File doesn't exist!\n");

} // open up the corresponfing file in read mode

if (file != NULL) {

// as long as the file exists read in all the values from the file

fscanf(file, "%d %d %d %d\n", &numPlayers, &numSnakesAndLadders, &boardRows, &boardCols);

// read in the non snakes/players/ladders info

players = (player\_t\*)malloc(sizeof(player\_t)\*numPlayers); // create array to store players

snakes = (snake\_t\*)malloc(sizeof(snake\_t)\*(numSnakesAndLadders)); // create array to store snakes

ladders = (ladder\_t\*)malloc(sizeof(ladder\_t)\*(numSnakesAndLadders)); // create array to store ladders

// re-create the arrays to store the players, snakes and ladders

gameBoard = (char\*\*)malloc(boardRows\*sizeof(char\*));

for (i = 0; i < boardRows; i++) {

gameBoard[i] = (char\*)malloc(boardCols \* sizeof(char));

}

// re-create board

boardSetUp(); // reset the contents of the board

for (i = 0; i < numSnakesAndLadders; i++) {

fscanf(file, "%d %d %d %d\n", &snakes[i].headCol, &snakes[i].headRow, &snakes[i].tailCol, &snakes[i].tailRow);

gameBoard[snakes[i].headRow][snakes[i].headCol] = 'S';

gameBoard[snakes[i].tailRow][snakes[i].tailCol] = 's';

// update position of the snakes

} // load snake info

for (i = 0; i < numSnakesAndLadders; i++) {

fscanf(file, "%d %d %d %d\n", &ladders[i].headCol, &ladders[i].headRow, &ladders[i].tailCol, &ladders[i].tailRow);

gameBoard[ladders[i].headRow][ladders[i].headCol] = 'L';

gameBoard[ladders[i].tailRow][ladders[i].tailCol] = 'l';

// update the board with the position of the ladders

}// load ladders info

for (i = 0; i < numPlayers; i++) {

fscanf(file, "%d %d %d\n", &players[i].col, &players[i].isTurn, &players[i].row);

} // load from file

fclose(file); // close the file

for (i = 0; i < numPlayers; i++) {

switch (players[i].isTurn) {

case 1: players[i].isTurn = true;

break;

case 0:

players[i].isTurn = false;

break;

default:

break;

} // assign each player their current "isTurn" value

}

printf("Game loaded !\n");

return true;

} else {

printf("Sorry, cannot opening the file.\n");

return false;

}

} // end of load