

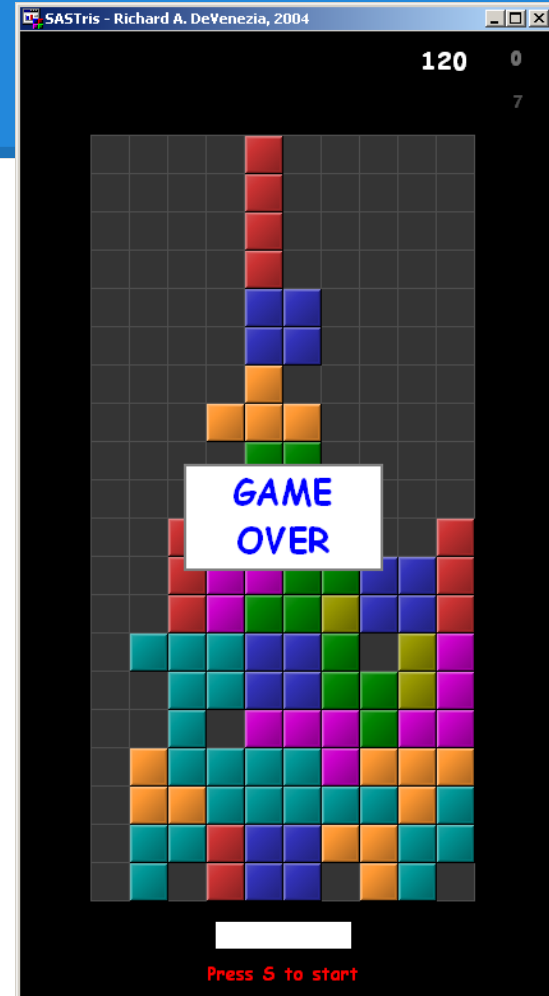
Tetris

---Michelle Neuburger, Kriti Shrestha, Yiran Zhang

Introduction:

Game Description: Tetris is a popular block game. The main components of the game are variously shaped tetrominos, which the player must stack on top of each other so as to create as few gaps as possible between the blocks. Once a row is completely filled with blocks, it is cleared away so as to make space for new tetrominos. The tetrominos are placed in a location specified by the player. The objective is to keep the game going for as long as possible without reaching the top-most row.

In other words, the player must try to clear as many rows as possible. Points are awarded for rows cleared.



Game design

Gameboard

`createGameboard`

`displayGameboard`

Tetromino

`constcharblock`

`randomSelectionOfTetromino`

`rotateTetromino`

`displayTetromino`

Score

`checkCompleted`

`clearCompleted`

`updateScore`

addTetro

`invalidMove`

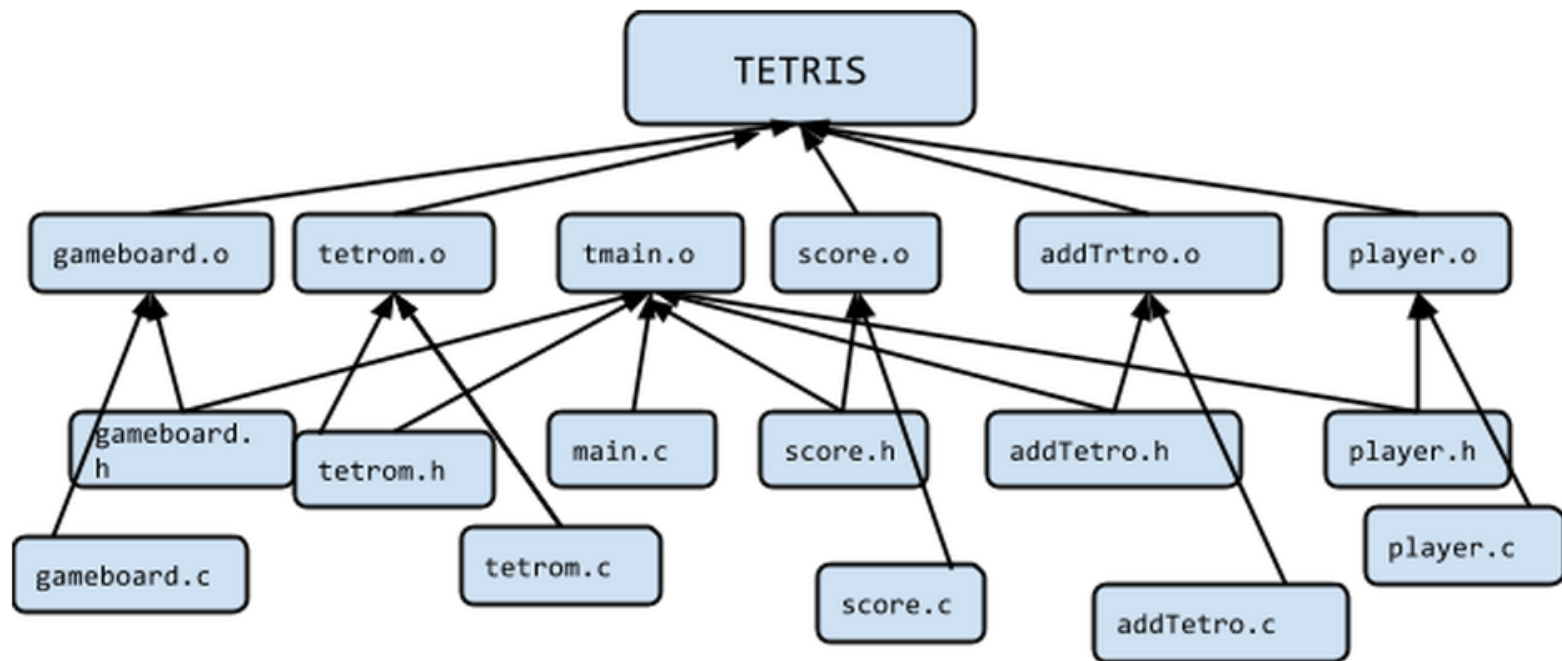
`addTetro`

`termination`

Player

`specifyLocation`

Dependency graph



Game design:

Tetromino

@@@@ @@ @ @@ @ @ @@
 @@ @@@ @@ @@@ @@@ @@

```
/* different shapes of tetrominos*/  
const char blockI[ 4 ][ 4 ] = { { ' ', '@', ' ', ' ', ' ', }, { ' ', '@', ' ', ' ', ' ', }, { ' ', '@', ' ', ' ', ' ', }, { ' ', '@', ' ', ' ', ' ', }, };  
const char blockJ[ 4 ][ 4 ] = { { ' ', '@', ' ', ' ', ' ', }, { ' ', '@', ' ', ' ', ' ', }, { '@', '@', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, };  
const char blockL[ 4 ][ 4 ] = { { '@', ' ', ' ', ' ', ' ', }, { '@', ' ', ' ', ' ', ' ', }, { '@', '@', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, };  
const char blockO[ 4 ][ 4 ] = { { '@', '@', ' ', ' ', ' ', }, { '@', '@', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, };  
const char blockS[ 4 ][ 4 ] = { { ' ', '@', '@', ' ', ' ', }, { '@', '@', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, };  
const char blockT[ 4 ][ 4 ] = { { '@', '@', '@', ' ', ' ', }, { ' ', '@', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, };  
const char blockZ[ 4 ][ 4 ] = { { '@', '@', ' ', ' ', ' ', }, { ' ', '@', '@', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, { ' ', ' ', ' ', ' ', ' ', }, };
```

Tetromino

```
void randomSelectionOfTetromino( )
{

    srand(time(NULL));
    char tBlock = ( rand( ) % 7 ) + 1;
    int i;
    int j;
    switch ( tBlock )
    {
        case 1: memcpy( tetroblock.tetro, blockI, 16 ); break;
        case 2: memcpy( tetroblock.tetro, blockJ, 16 ); break;
        case 3: memcpy( tetroblock.tetro, blockL, 16 ); break;
        case 4: memcpy( tetroblock.tetro, blockO, 16 ); break;
        case 5: memcpy( tetroblock.tetro, blockS, 16 ); break;
        case 6: memcpy( tetroblock.tetro, blockT, 16 ); break;
        case 7: memcpy( tetroblock.tetro, blockZ, 16 ); break;
    }
    //tetroblock.tetroX = 6;
    //tetroblock.tetroY = 0;
    displayTetromino(tetroblock.tetro);
}
```

Rotate

```
void rotateTetromino( )
{
    char newTetro[ 4 ][ 4 ];
    int i;
    int j;
    for ( i = 3; i >= 0; --i )
        for ( j = 0; j < 4; ++j )
            newTetro[ j ][ 3-i ] = tetroblock.tetro[ i ][ j ];

    //for ( i = 0; i < 4; ++i )
    //for ( j = 0; j < 4; ++j )
    //newTetro[ i ][ j ] = tetroblock.tetro[ j ][ i ];
    displayTetromino(newTetro);
}

void displayTetromino(char t[][4])
{
    int i;
    int j;

    for (i=0; i<4; i++)
    {
        for (j=0; j<4; j++)
            printf("%c", t[i][j]);
        printf("\n");
    }
}
```


Specify location

```
int specifyLocation()
{
do{
printf("Enter the column number (0 - %d) where you want the leftmost block of the tetromino: ", MAXCOL);
scanf("%d", &col);
printf("You entered %d.\n", col[0]);
if (col[0]>MAXCOL)
{
printf("Invalid input!\n");
specifyLocation();
}
return col[0];} while (col[0]>MAXCOL);
}
```

Addtetro

```
int addTetro(int rows, int cols, char board[][cols], char tetra[][4], int desig) {
//returns 1 if successfully added, 0 if unsuccessful
int signal1 = 0;
int signal2 = 0;
int mincol;
int maxcol;
int minrow;
int startcol;
int startrow;
int i, j, k;
int heights[4] = {0};
for (i = 3; i >= 0; i--) {
for (j = 0; j < 4; j++) {
if (tetra[i][j] == '@'){
if (signal1 == 0) {
minrow = i;
maxcol = j;
mincol = j;
startrow = i;
startcol = j;
signal1 = 1;
} else {
if (i < minrow) {
minrow = i;
}
if (j < mincol) {
mincol = j;
} else if (j > maxcol) {
maxcol = j;
}
```

```

signal1 = 0; //no collisions
signal2 = 0; //lowest working row not found yet
//printf("Columns: %d to %d, Rows: %d to %d\n", mincol, maxcol, minrow, startrow);
int lcoloff = startcol - mincol;
int rcoloff = maxcol - startcol;
int tetracols = maxcol-mincol + 1;
int tetrarows = startrow-minrow + 1;
int bsrow;
//printf("Offsets:\nleft: %d\tright: %d\nrows: %d\trcols: %d\n", lcoloff, rcoloff, tetrarows, tetracols);

for (k = rows-1; k >= 0; k--) {
//printf("%d\n", k);
if (signal2 == 0) {
for (i = startrow; i >= minrow; i--) {
for (j = mincol; j <= maxcol; j++) {
int browoffset = k - (startrow - i);
int bcoloffset = desig - lcoloff + (j - mincol);
int throwffset = i;
int tcoloffset = j;
if (browoffset < 0) {
return youLose();
}
if ((bcoloffset < 0) || (bcoloffset > cols-1)) {
return invalidMove(desig);
}
char b = board[browoffset][bcoloffset];
char t = tetro[throwffset][tcoloffset];
if ((b == '@') && (t == '@')){
signal1 = 1;
}
}
}
}

```

Addtetro(cont)

```
if (signal1 == 0) {
    bsrow = k;
    signal2 = 1;
}
}
signal1 = 0;
}
if (signal2 == 0) {
    return youLose();
} else {
    for (i = tetrarows; i > 0; i--) {
        int broffset = bsrow - (tetrarows - i);
        int troffset = startrow - (tetrarows - i);
        for (j = mincol; j <= maxcol; j++) {
            int bcoffset = desig - lcoloff + (j - mincol);
            int tcoffset = startcol - lcoloff + (j - mincol);
            char b = board[broffset][bcoffset];
            char t = tetra[troffset][tcoffset];
            //printf("(%d, %d) %c (%d, %d) %c\n", broffset, bcoffset, b, troffset, tcoffset, t);
            if ((b != '@') && (t == '@')){
                board[broffset][bcoffset] = tetra[troffset][tcoffset];
            }
        }
    }
}
return 1;
}
```

Score

Check complete row

```
int checkCompleted(const int rows, const int cols, char board[][cols]) {  
    //ASSUMES EMPTY SPACE IS ' ' AND FULL SPACE IS '@'. MODIFY CHARS IF NECESSARY.  
    int signal = 0;  
    for (int i = 0; i < rows; i++) {  
        for (int j = 0; j < cols; j++) {  
            if (board[i][j] == ' ') {  
                signal = 1;  
            }  
        }  
        if (signal == 0){  
            return i;  
        }  
        signal = 0;  
    }  
    return size+1;  
}
```

Score

Clear complete row

```
int clearCompleted(int rows, int cols, char board[][cols], int compidx){
//ASSUMES EMPTY SPACE IS ' ' AND FULL SPACE IS '@'. MODIFY CHARS IF NECESSARY.
if (compidx > rows) {
    return 0;
}
for (int i = compidx; i > 0; i--) {
    for (int j = 0; j < cols; j++) {
        board[i][j] = board[i-1][j];
    }
}
for (int j = 0; j < cols; j++){
    board[0][j] = ' ';
}
return 1;
}
```

Score

Update score

```
void updateScore(int * score, int signal) {  
    /*if the signal is a 1, add 10 to the score.  
    otherwise do nothing. */  
    if (signal > 0) {  
        *score += 10;  
    }  
}  
  
/*so the above would be used like: int r = checkCompleted(n, board);  
while (r < n+1) {  
    updateScore(score, clearCompleted(n, board, r));  
    r = checkCompleted(n, board);  
} */
```