Tetris Game

PRESENTED BY

IIT2022119 - TANAY FALOR

IIT2022156 - NIRBHAY PALIWAL

IIT2022157 - ADITYA BHARDWAJ

IIT2022158 - SHIVAM KUMAR

IIT2022160 - MALAY SAHITYA

IIT2022162 - RAVI PRAKASH

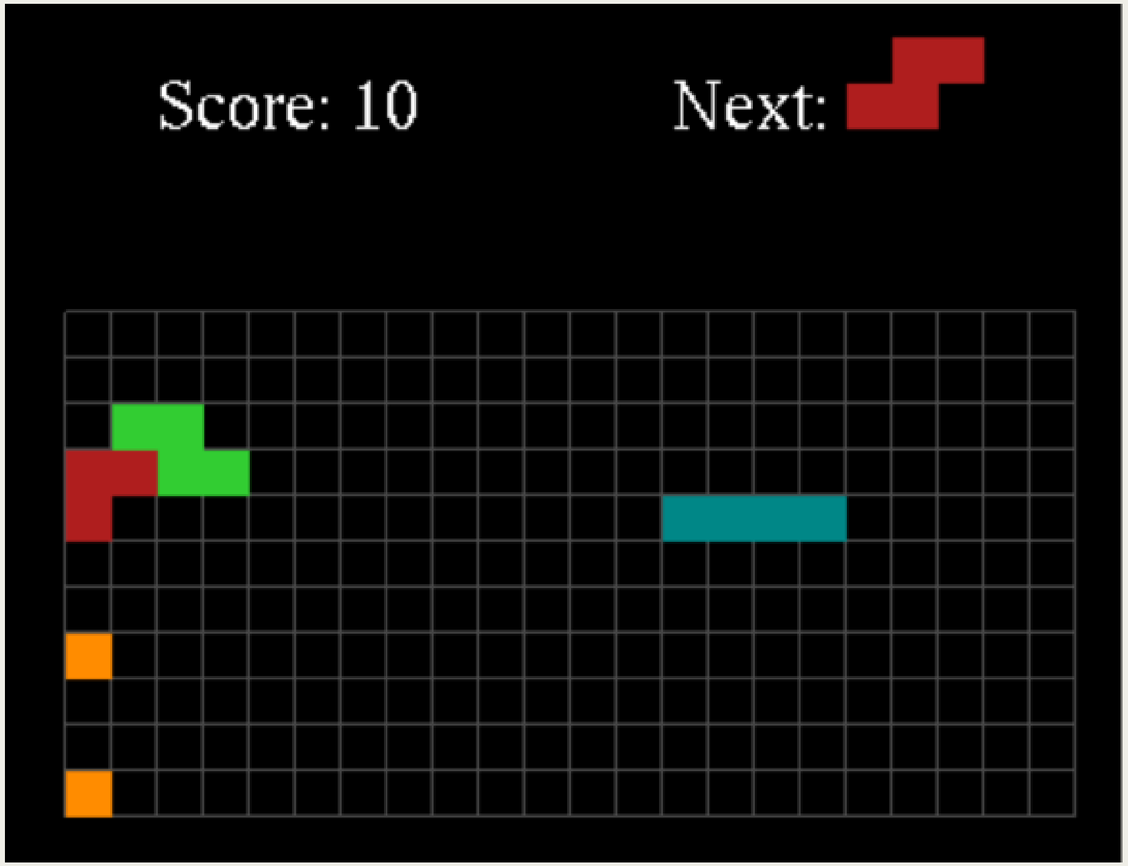
IIT2022205 - PRATEEK SARVAIYA

COURSE INSTRUCTOR

PAVAN SIR &

ANJALI MA'AM

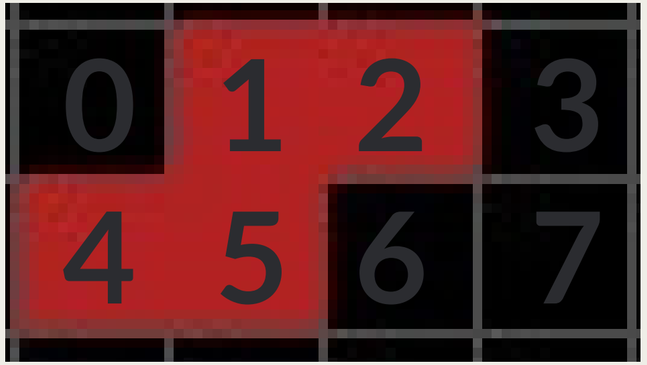
INTRODUCTION

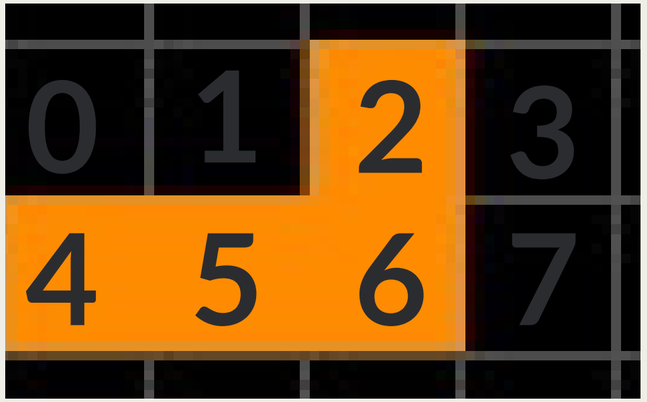


- Developed in OpenGL and C++.
- The game features falling blocks of different shapes, which the player can move from up and down and rotate to fit into columns.

Game Variables

- Game uses a 2D field represented by a grid (field[M][N]).
- a: This stores the next position of current shape.
- b: This stores the current position.
- c: This stores the next shape that will replace the current one.






```
#define B 0  // Vanish zone height
#define M 11 // Field height
#define N 22 // Field width
const float S = 0.05; // Square size
const int figures[7][4] = {
    {4, 5, 6, 7}, // I
    {1, 2, 5, 6}, // O
    {4, 5, 6, 1}, // T
    {0, 1, 5, 6}, // S
    {4, 5, 1, 2}, // Z
    {0, 4, 5, 6}, // J
    {4, 5, 6, 2}, // L
}; /* Figure matrix
 * 0 1 2 3
 * 4 5 6 7
 */
const float colors[7][3] = {
    {0.000, 0.545, 0.545}, // Cyan
    {1.000, 0.843, 0.000}, // Yellow
    {0.545, 0.000, 0.545}, // Purple
    {0.196, 0.804, 0.196}, // Green
    {0.698, 0.133, 0.133}, // Red
    {0.098, 0.098, 0.439}, // Blue
    {1.000, 0.549, 0.000}, // Orange
};
struct point {
    int x, y;
} a[4], b[4], c[4];

int delay = 600;
int field[M][N] = {0};
int shape, shapeNext;
int gameOver = 0;
```

Initialization and Setup

- glClearColor(0.0, 0.0, 0.0, 1.0): This sets the background color to black.
- srand(time(NULL)): Initializes the random number generator with the current time, ensuring different sequences of random numbers on each run.
- generateFigure() and spawnFigure(): These functions are called to create and spawn the next block or "figure" that will fall in the game.
- Registers draw() for rendering, timerFunc() for periodic updates (game loop), and keyboardFunc() for handling key events.
- The program enters the GLUT event loop with glutMainLoop(), which keeps the game running.

```
void init(void) {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glEnable(GL_DEPTH_TEST);

    srand(time(NULL));

    generateFigure();
    spawnFigure();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(640, 640);
    glutCreateWindow("Tetris");

    init();
```

```
    glutDisplayFunc(draw);  // screen needs to be updated (e.g., after resizing the window or explicitly
                            // calling glutPostRedisplay())
    glutTimerFunc(delay, timerFunc, 0); // After delay milliseconds, the timerFunc function is called.
    glutKeyboardFunc(keyboardFunc);// Whenever a key is pressed, OpenGL will call the keyboardFunc function.
    glutMainLoop();

    return 0;
}
```

Check() Function

- Preventing Out-of-Bounds Movement.

- Collision Detection: It ensures that the active piece doesn't overlap with already-placed blocks, which could cause the game to break or place blocks in impossible positions.

```
int check(void) {
    for (int i = 0; i < 4; i++) {
    // Ensure the block stays within the horizontal bounds of the field
        if (a[i].x < 0 || a[i].x >= N) return 0;
    // Ensure the block stays within the vertical
    // bounds of the field, including above the vanish zone
        if (a[i].y < 0 || a[i].y >= M) return 0;
    // Ensure the block does not collide with existing blocks
        if (field[a[i].y][a[i].x]) return 0;
    }
    return 1;
}
```

Timer Function

- Move Piece Left

  b array is used to store

  previous result

```
for (int i = 0; i < 4; i++) {
    b[i] = a[i];
    a[i].x -= 1;  // Move left
}
```

- Collision Detection

  Checking if incoming block collides

  with existing blocks .It is checked

  using check function which is checking

  out of bound condition.

```
if (!check()){
    for (int i = 0; i < 4; i++){
        field[b[i].y][b[i].x] = shape + 1;
    }
    clearColumn();
    if (!gameOver) {
        spawnFigure();
    }
}
```

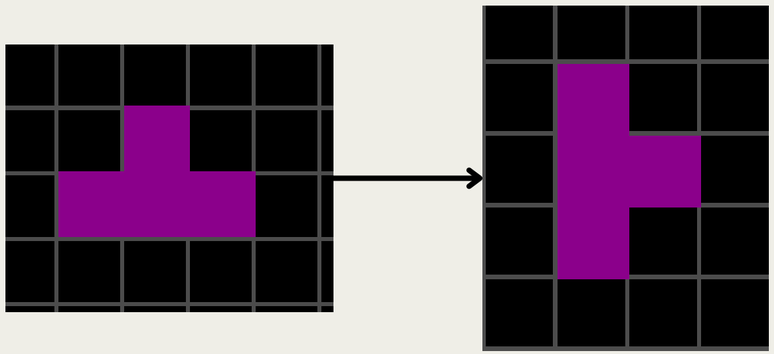
Timer Function

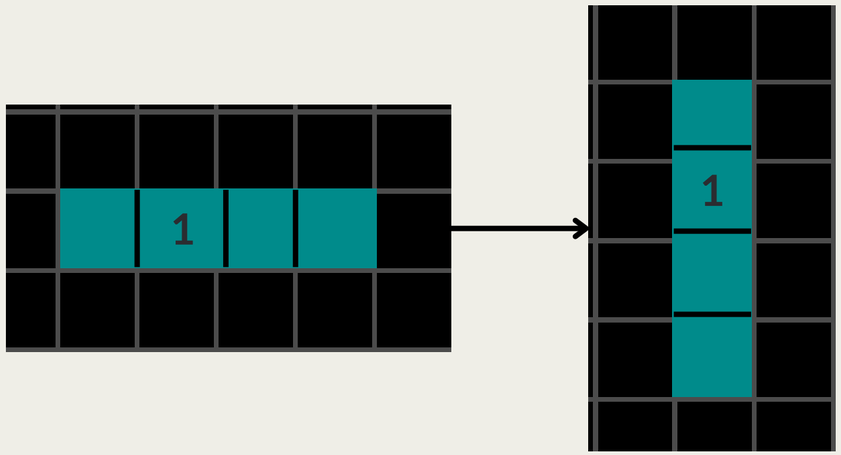
- Column Clearance

  If any of the cell of leftmost column has not been visited then we will not clear leftmost column.

```
void clearColumn(void) {

    for (int j = 0; j < N; j++) {
        int fullColumn = 1;
        for (int i = B; i < M; i++) {
            if (field[i][j] == 0) {
                fullColumn = 0;
                break;
            }
        }

        if (fullColumn) {
            score += 10;
            for (int i = B; i < M; i++) {
                for (int k = j; k < N - 1; k++) {
                    field[i][k] = field[i][k + 1];
                }
                field[i][N - 1] = 0;
            }
        }
    }

}
```

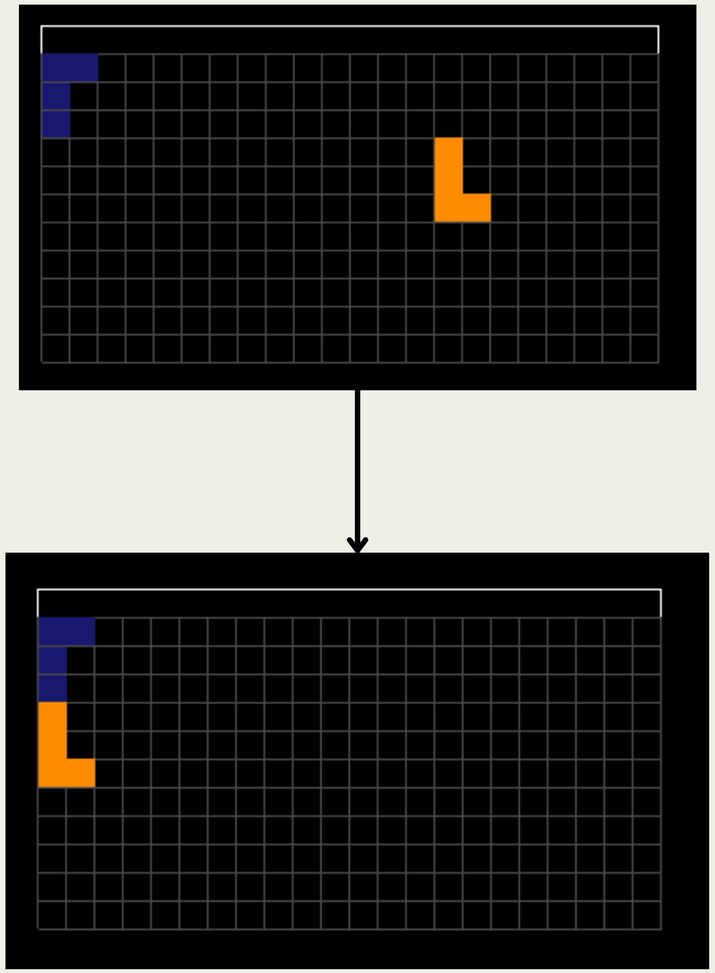
ROTATE






```
void rotate(void) {
    // Rotation around a[1]
    point p = a[1];
    for (int i = 0; i < 4; i++) {
        int x = a[i].y - p.y;
        int y = a[i].x - p.x;

        a[i].x = p.x - x;
        a[i].y = p.y + y;
    }
    // Reset the figure if it's out of bounds
    if (!check()) {
        for (int i = 0; i < 4; i++) {
            a[i] = b[i];
        }
    }
}
```

DROP



```
void drop() {
    while(1)
    {
        for (int i = 0; i < 4; i++) {
        b[i] = a[i];
        a[i].x -= 1;
        }
        if (!check()) {
            for (int i = 0; i < 4; i++) {
                a[i] = b[i];
            }
            break;
        }
    }
}
```

Draw Function

- The function visualizes the game state, including the current figure, the next figure, the grid, and messages such as "Game over."
- glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT): Clears the screen and depth buffers.
- glLoadIdentity(): Resets the transformation matrix to its default state.
- gluOrtho2D(-1.0, 1.0, 1.0, -1.0): Sets up a 2D orthogonal projection with inverted Y-axis to match a Cartesian coordinate system.
- And than Draw the current Figure.
- And than if Game not over generate the next figure and score.

```
// Start drawing
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();

// Invert Y axis
gluOrtho2D(-1.0, 1.0, 1.0, -1.0);
```

```
float Ox = 1.0 - N * S * 0.5; // Offset for the game field
float Oy = 1.0 - M * S * 0.5;
// Draw the current figure
for (int i = 0; i < 4; i++) {
    // Draw all blocks in the figure, regardless of their position
    setFigureColor(shape);
    drawRectangle(a[i].x * S, a[i].y * S, S, S, Ox, Oy, 0);
}
```

Draw Function

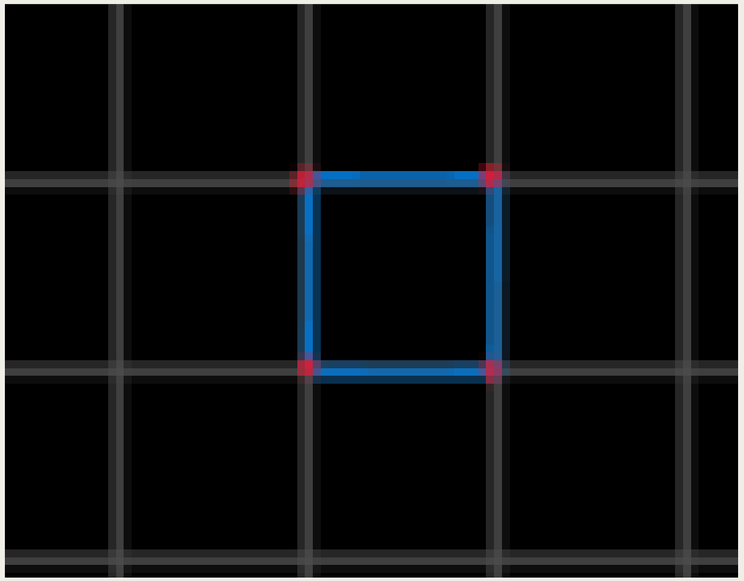
- After drawing the next figure draw the feild squares on their positions.
- If game is over than display the text that game is over .

```
// Draw the field squares
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        if (field[i][j] != 0) {
            setFigureColor(field[i][j] - 1);
            drawRectangle(j * S, i * S, S, S, Ox, Oy, 0);
        }
        // Draw the grid
        glColor3f(0.3, 0.3, 0.3);
        drawRectangle(j * S, i * S, S, S, Ox, Oy, GL_LINE_LOOP);
    }
}
```

```
// Game over message
if (gameOver) {
    glColor3f(1.0, 0.3, 0.0);
    drawText(0, 0, (char *)"Game over!", Ox + N * S * 0.5 - 2 * S, Oy + 0.2 * S, NULL);
}
```

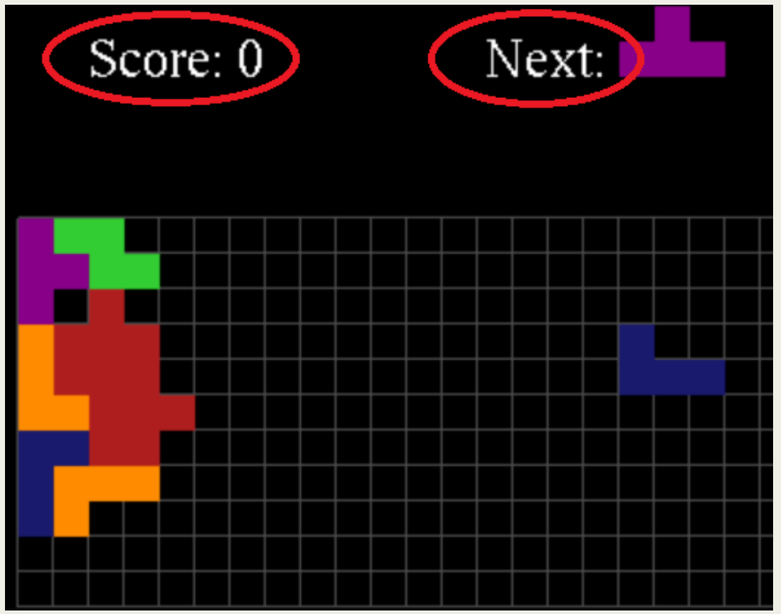
Drawing Rectangle Boxes

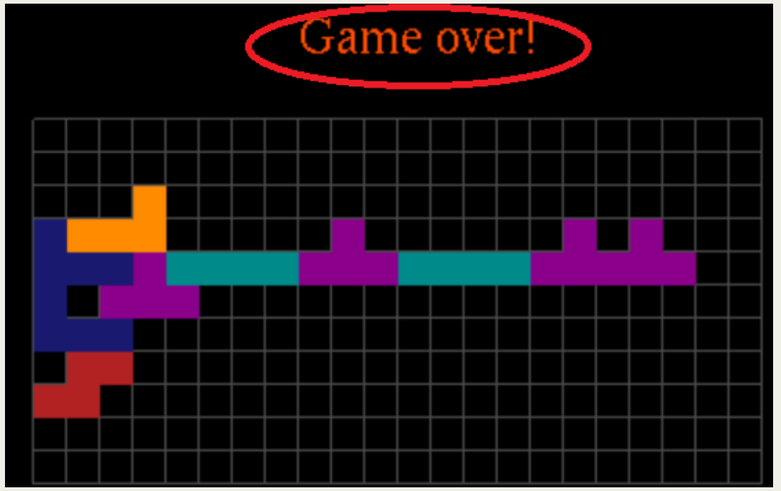
```
void drawRectangle(float x, float y, float width, float height,
                   float Ox, float Oy, GLenum type) {
    if (type == 0) type = GL_POLYGON;

    //(x, y) -> Top-left corner
    y = -1.0 + y + Oy;
    x = -1.0 + x + Ox;

    glBegin(type);
    glVertex2f(x, y - height);
    glVertex2f(x + width, y - height);
    glVertex2f(x + width, y);
    glVertex2f(x, y);
    glEnd();
}
```



Writing Text

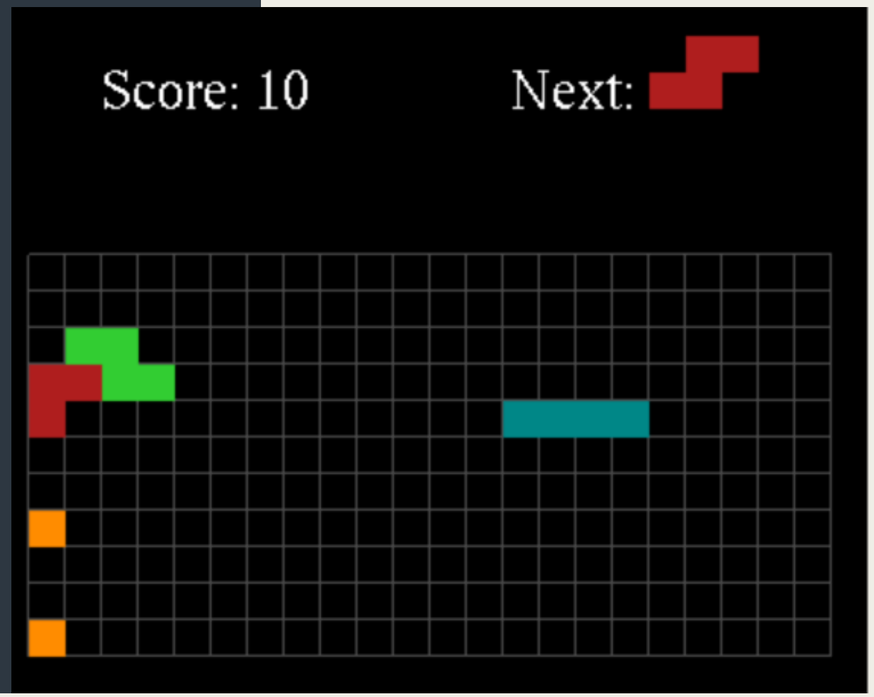
```
void drawText(float x, float y, string s,
                    float Ox, float Oy, void *font){

  if (font == NULL) font = GLUT_BITMAP_TIMES_ROMAN_24;

  y = -1.0 + y + Oy - 0.1;
  x = -1.0 + x + Ox + 0.2;

  glRasterPos2f(x, y);
  int len = s.length();
  for(int i=0;i<len;i++){
    glutBitmapCharacter(font, s[i]);
  }
```





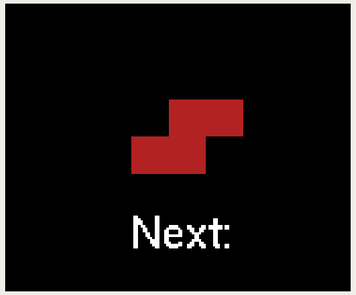
Showing Score

```
void drawScore(float x, float y, int num, float Ox, float Oy, void *font) {
  if (font == NULL) font = GLUT_BITMAP_TIMES_ROMAN_24;

  y = -1.0 + y + Oy;
  x = -1.0 + x + Ox;

  string s = "";
  if(num == 0) s += '0';
  while(num > 0){
    int d = num%10;
    num /= 10;
    s += (char)(d+'0');
  }
  int len = s.length();
  for(int i=0;i<len/2;i++) swap(s[i],s[len-i-1]);

  glRasterPos2f(x, y);
  for(int i=0;i<len;i++){
    glutBitmapCharacter(font, s[i]);
  }
}
```



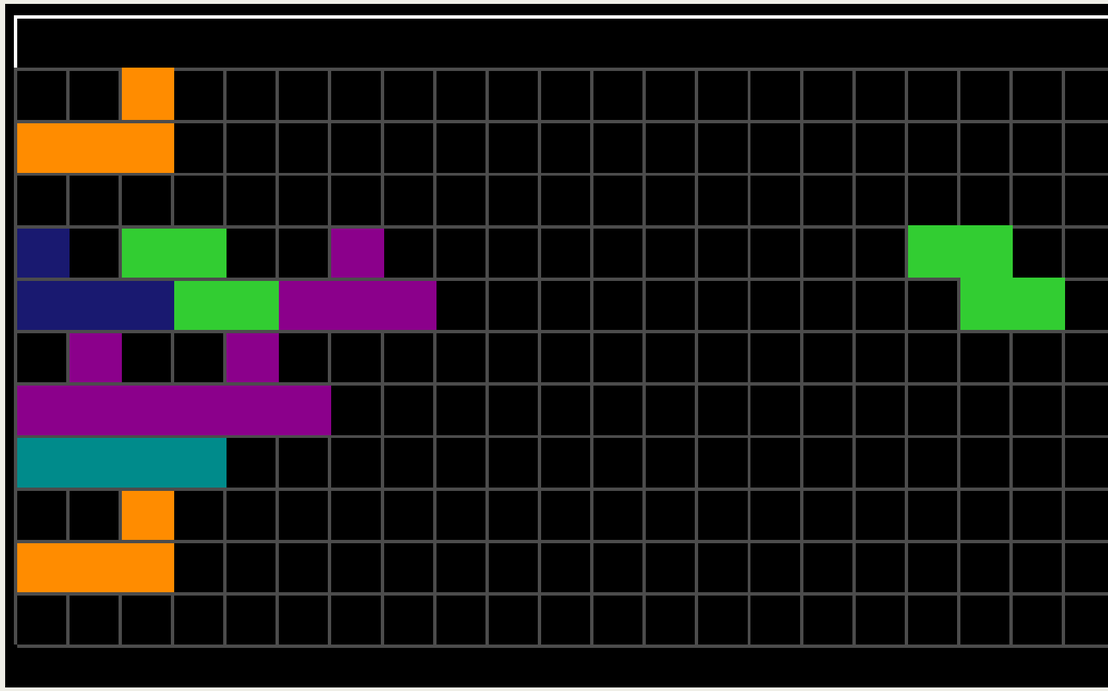
Generate Figure

```
void generateFigure(void) {

    shapeNext = rand() % 7;

    for (int i = 0; i < 4; i++) {
        c[i].x = figures[shapeNext][i] % 4 ;
        c[i].y = figures[shapeNext][i] / 4;
    }
}
```



Spawn Figure

```
void spawnFigure(void) {
    shape = shapeNext;

    for (int i = 0; i < 4; i++) {
        a[i].x = figures[shape][i] % 4 + N - 4; // Start near the right edge
        a[i].y = figures[shape][i] / 4 + M * 0.5 - 2; // Center vertically
    }

    for (int i = 0; i < 4; i++) {
        b[i] = a[i];
        a[i].x -= 1; // Move one square left to start
    }

    if (!check()) {
        gameOver = 1;
    }

    generateFigure();
}
```



Interacting using Keyboard

- **key: 27**: To quit (ESC)
- **u**: To move current block in up direction
- **d**: To move current block in down direction
- **a**: Immediate move block to valid rightmost position
- **key: 32**: To rotate the block (Spacebar)

```
void keyboardFunc(unsigned char key, int x, int y) {
    switch (key) {
        case 27:  // ESC
            exit(0);
            break;

        case 'u':  // Move up
            for (int i = 0; i < 4; i++) {
                b[i] = a[i];
                a[i].y -= 1;
            }
```

Interacting using Keyboard

- The **check()** function ensures the new position is valid
  - If not **revert the changes**
- After processing a key press, the **glutPostRedisplay()** function is called to update the display
  - Ensuring the visual representation matches the new state.

```
// Ensure the figure doesn't move out of bounds or collide
if (!check()) {
    for (int i = 0; i < 4; i++) {
        a[i] = b[i];
    }
}
break;
```