Computer Graphics (UCS505)

AstroBlasters: Galactic Showdown

B.E. 3rd Year CSE (3Q33)

Submitted by:

Natesh Singh Pundir (102216077)
Kushal Yadav (102216049)
Sahil Rashid (102216094)

Submitted To – Dr.

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

INTRODUCTION

Project Overview:

This project is a **multi-level asteroid shooting game** developed using **OpenGL** and **C++** with the GLUT toolkit. The game mimics the classic *Asteroids* arcade gameplay, where a player controls a rotating spaceship that can shoot bullets to destroy moving asteroids.

Scope of the Project:

✅ Current Scope

1. **Core Gameplay Mechanics**:
    - Player-controlled spaceship with rotation and shooting capabilities.
    - Finite bullets per level (adds challenge and strategy).
    - Collision detection between bullets and asteroids.
2. **Multi-Level Progression**:
    - 10 game levels with increasing asteroid count and difficulty.
    - Score tracking to reward performance.
    - Game over logic for both success (all levels completed) and failure (out of bullets).
3. **Dynamic Environment**:
    - Randomly generated asteroids each level (position, color, speed, shape).
    - Screen wrapping for asteroids to maintain continuous motion.
    - Basic UI (score, level, bullets left) and visual modes (grid/pattern).
4. **User Interface**:
    - Text-based UI using GLUT bitmap fonts.
    - Menu interaction for toggling background styles.
5. **Graphics and Rendering**:
    - 2D rendering using OpenGL and GLUT.
    - Basic animation and rendering loop with `glutTimerFunc()`.

🔮 Future Scope (Possible Enhancements)

1. **Gameplay Features**:
    - Power-ups (extra bullets, shields, multi-shot).
    - Asteroid splitting (larger ones break into smaller ones).
    - Health bar for the ship or lives system.
2. **Graphics & Effects**:
    - Add explosion effects and sound on collision.
    - Improve visual aesthetics using textures or shaders.
3. **Input & Controls**:
    - Support for mouse or gamepad input.
    - Movement-based control instead of rotation-only.
4. **Advanced Collision Detection**:
    - More precise polygonal or pixel-based collision.
    - Ship-asteroid collision logic (currently only bullets destroy asteroids).
5. **High Score System**:
    - Store and display high scores.
    - Include a main menu, pause/resume, and restart features.
6. **Portability & Platform**:
    - Port to mobile or web (e.g., using WebGL or SDL).
    - Refactor using modern OpenGL for better cross-platform performance.
7. **Multiplayer / AI**:
    - Introduce AI enemies or co-op gameplay.

📌 Conclusion

This project serves as a solid foundation for a 2D arcade-style game using OpenGL. It effectively demonstrates game design principles, procedural content generation, and animation logic. While basic in scope, it offers **numerous opportunities for expansion** into a more complete and engaging game.

wUSER DEFINED FUNCTIONS

S No.	Function Name	Function Description
1	**Void Shoot()**	The **`shoot()`** function is a key feature in the game, allowing the player to fire bullets from the spaceship.
2	**Void drawScore(void *front)**	The **`drawScore()`** function is responsible for rendering the player's score on the screen. It displays the current score in the game
3	**Void drawship()**	The **`drawShip()`** function is responsible for rendering the spaceship (or player ship) in the game. It uses OpenGL to draw the ship, which is a simple triangular shape, and rotates it based on the player's current orientation
4	**Void drawAsteroid(int)**	**The drawAsteroid(int i) function is responsible for rendering an asteroid in the game at a specific position and with a certain appearance. Each asteroid has a random color, size, and number of sides, giving it a unique and unpredictable appearance.**
5	**Void drawBullets(int)**	The **`drawBullet(int i)`** function is responsible for drawing a bullet on the screen at its current position. It renders the bullet as a simple line that represents its trajectory.
6	**Void drawLevel()**	The **`drawLevel(void* font)`** function is responsible for displaying the current **game level** on the screen. This helps the player keep track of the progress throughout the game.
7	**Void drawGameOver()**	The `drawGameOver` function is responsible for rendering the "Game Over" message when the game ends.
8	**Void updateAsteroids**	The `updateAsteroids()` function is responsible for updating the positions of all the asteroids in the game during each frame. It handles asteroid movement and screen wrapping, ensuring that asteroids stay within the game boundaries by wrapping around the screen when they move off the edge.
9	**Void randomizedAsteroids()**	The `randomizeAsteroid()` function is responsible for creating new asteroids at random positions within the game screen.
10	**Void detectCollision()**	The `detectCollision()` function is designed to detect when an object in the game (such as the player's bullets or the ship) collides with an asteroid.

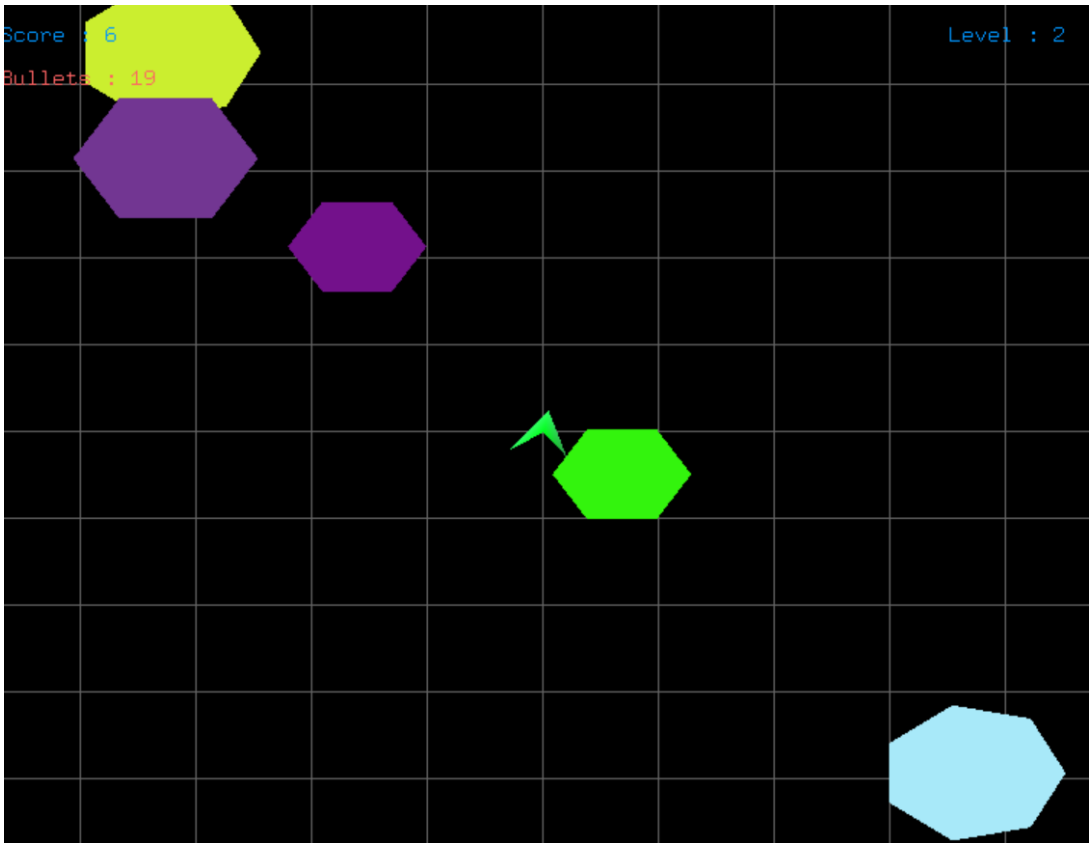
CODE SNIPPETS

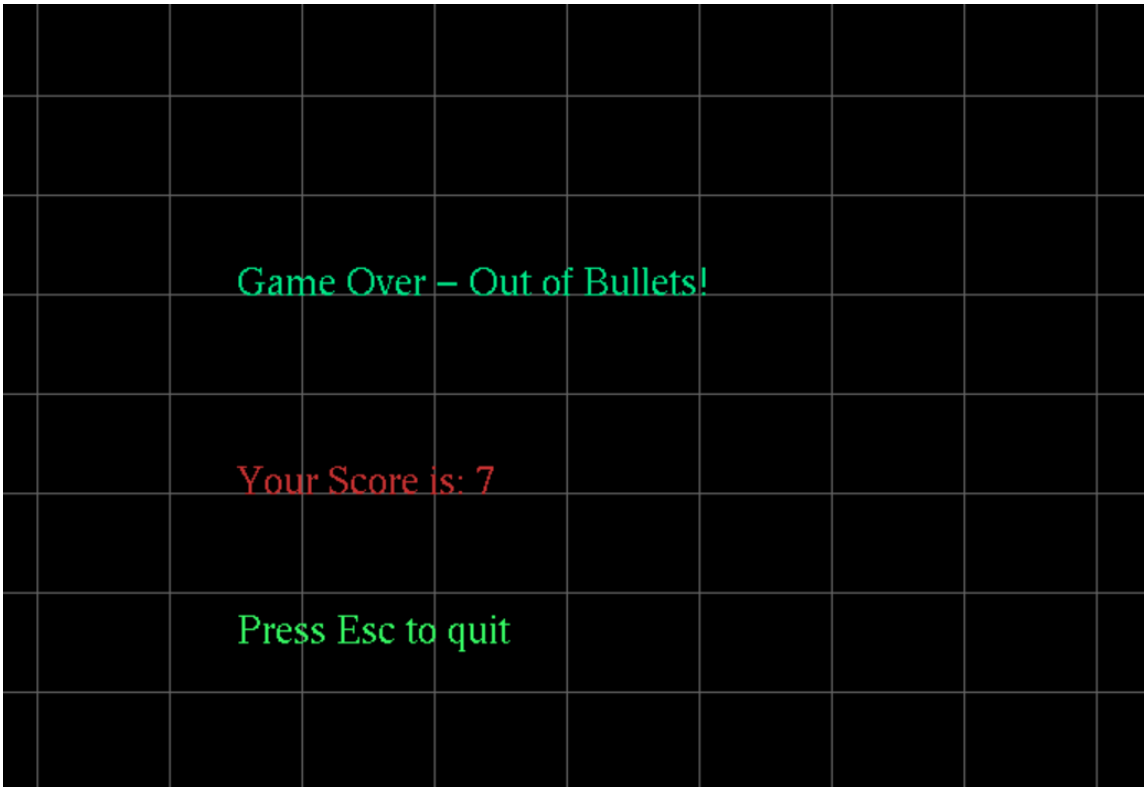
```
void drawAsteroid(int i) {
    glLoadIdentity();
    glTranslated(asteroids[i]->centerX, asteroids[i]->centerY, 0.0);
    glColor3ub(asteroids[i]->red, asteroids[i]->green, asteroids[i]->blue);
    glBegin(GL_POLYGON);
    for (int j = 0; j < asteroids[i]->numOfSides; j++) {
        double theta = 2.0 * PI * double(j) / double(asteroids[i]->numOfSides);
        glVertex2d(asteroids[i]->radius * cos(theta), asteroids[i]->radius * sin(theta));
    }
    glEnd();
}

void drawBullet(int i) {
    glLoadIdentity();
    glColor3ub(100, 255, 140);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    glVertex2d(bullets[i]->x, bullets[i]->y);
    glVertex2d(bullets[i]->x + bullets[i]->movementX, bullets[i]->y + bullets[i]->movementY);
    glEnd();
    bullets[i]->x += bullets[i]->movementX;
    bullets[i]->y += bullets[i]->movementY;
    // Remove bullet if out of bounds
    if (bullets[i]->x < -55 || bullets[i]->x > 55 || bullets[i]->y < -55 || bullets[i]->y > 55) {
        delete bullets[i];
        bullets[i] = nullptr;
    }
}

void drawScore(void* font) {
    string scoreString = "Score : " + to_string(score);
    glColor3ub(0, 150, 255);
    glRasterPos2f(-47, 45);
    for (int i = 0; i < scoreString.length(); i++) {
        glutBitmapCharacter(font, scoreString[i]);
    }
}

void drawLevel(void* font) {
    string levelString = "Level : " + to_string(level);
    glColor3ub(0, 150, 255);
    glRasterPos2f(35, 45);
    for (int i = 0; i < levelString.length(); i++) {
        glutBitmapCharacter(font, levelString[i]);
    }
}
```

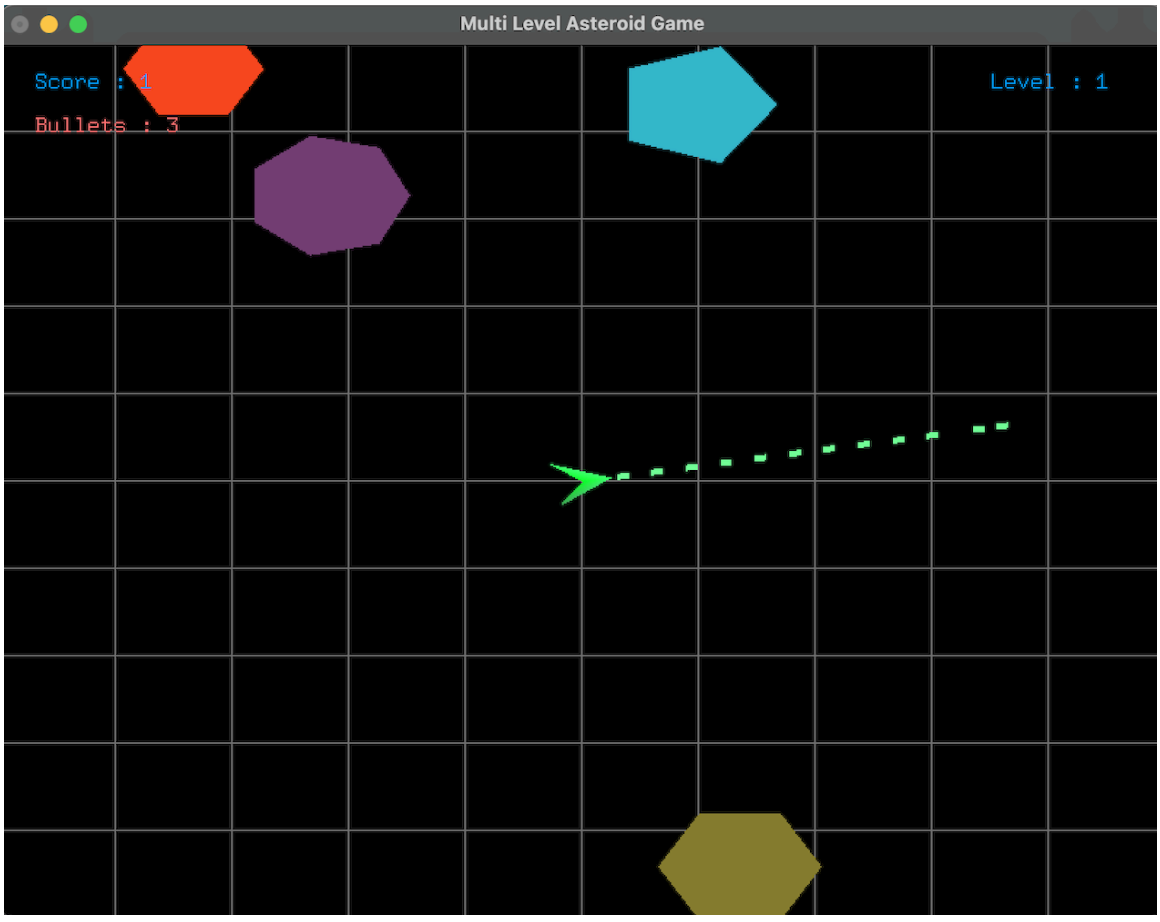
```
void drawShip() {
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glPushMatrix();
    glRotatef(angle, 0.0, 0.0, 1.0);
    glBegin(GL_POLYGON);
    glColor3ub(0, 255, 25);
    glVertex2f(0.0, 0.0);
    glColor3ub(50, 150, 75);
    glVertex2f(-2.5, -2.5);
    glColor3ub(50, 255, 100);
    glVertex2f(2.5, 0.0);
    glVertex2f(-2.5, 2.5);
    glEnd();
    glPopMatrix();
}

void drawBulletsLeft(void* font) {
    string bulletsString = "Bullets : " + to_string(bulletsLeft);
    glColor3ub(255, 100, 100);
    glRasterPos2f(-47, 40);
    for (int i = 0; i < bulletsString.length(); i++) {
        glutBitmapCharacter(font, bulletsString[i]);
    }
}

//Asteroid Movement
void updateAsteroids() {
    for (int i = 0; i < levelAsteroids; i++) {
        if (asteroids[i] != nullptr) {
            asteroids[i]->centerX += asteroids[i]->velocityX;
            asteroids[i]->centerY += asteroids[i]->velocityY;
            // Screen wrapping
            if (asteroids[i]->centerX > 50) asteroids[i]->centerX = -50;
            if (asteroids[i]->centerX < -50) asteroids[i]->centerX = 50;
            if (asteroids[i]->centerY > 50) asteroids[i]->centerY = -50;
            if (asteroids[i]->centerY < -50) asteroids[i]->centerY = 50;
        }
    }
}
```

();

SCREENSHOTS




Game Over – Out of Bullets!
Your Score is: 7
Press Esc to quit


Multi Level Asteroid Game
Score : 1
Bullets : 3
Level : 1

References

- OpenGL Utility Toolkit (GLUT) – Used for windowing, input handling, and rendering:https://www.opengl.org/resources/libraries/glut/

- OpenGL API Documentation – For rendering 2D/3D graphics:https://www.khronos.org/registry/OpenGL-Refpages/gl4/

- C++ Standard Library (iostream, string, etc.) – For input/output and string handling:https://cplusplus.com/reference/

- Trigonometric Functions (math.h) – For computing movement and rotation:https://cplusplus.com/reference/cmath/

- Inspiration and Concepts from Classic Games – The game mechanics and levels are inspired by classic arcade games like *Asteroids*.