3019 Report

Asteroid Astronaut

Gameplay

The game asteroid astronaut runs in the terminal and uses ASCII formatting to make up the graphics. The game is influenced by classic asteroid shooter games and follows early mobile/browser flash game principles remaining simple but difficult. The player must move the spaceship (>) around the stage destroying as many asteroids (@) while ensuring none of them hit the player otherwise they will lose a life, the game starts off simple but becomes overwhelming after some time. The goal of the game is to obtain the highest score possible with the game tracking and ordering the highest 5 scores while displaying them at all times.

Dependencies

Asteroid Astronaut utilises standard C++ libraries

- *<iostream> - console input/output and rendering*
- *<vector> -managing collections*
- *<cctype> -case insensitive input handling*
- *<conio.h>*
- *<windows.h>*
- *<algorithm> -for vector operations*
- *<iomanip> -text alignment*
- *<fstream> -reading/writing highscores*

AI implementation

AI (Grok) was used as an assistive tool throughout development. AI was used to provide code assistance, particularly for debugging one scenario where AI accelerated the debugging process by trying to get the highscores system working. The implementation of this system took a while and I made plenty of simple mistakes doing this ranging from file loading errors to simple display errors using '<iomanip>'. Using AI, I was able to find these errors much quicker and get the Highscore system implemented at a much faster rate.

Beyond debugging I used AI as a mentor for C++ and OOP practices and used it when I was ever stuck or unsure. I often shared code sections for review and received suggestions; however, all prompts were never applied blindly and I ensured any recommendations aligned with my original design goals and I understood how I was applying any snippets of code.

In summary:

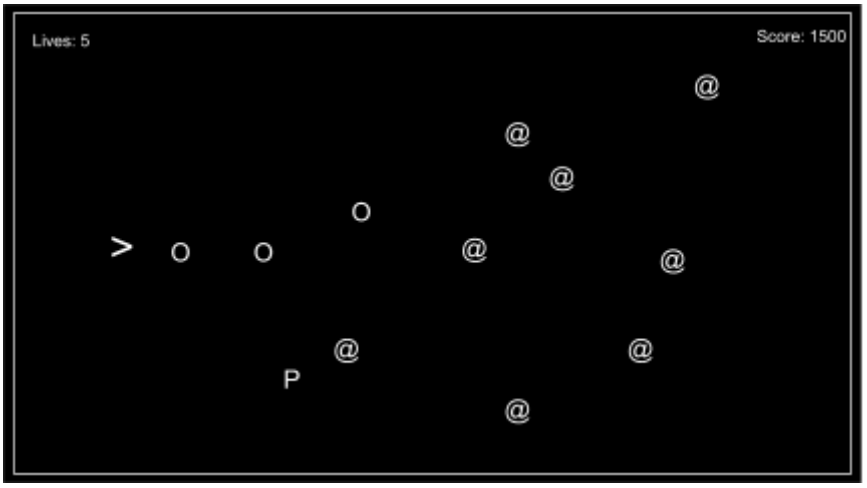
- *Ai accelerated debugging*
- *Deepened C++ understanding*
- *I remained the lead developer and didn't follow AI blindly*
- *Final implementation is my own with AI serving as a tool*

Testing

Test Cases	***Description***	***Expected***	***Actual***	***Pass?***
Movement	*Pressing WASD*	*Player moves* *Up (W)* *Down (S)* *Left (A)* *Right (D)*	*Player moves* *Up (W)* *Down (S)* *Left (A)* *Right (D)*	*Yes*
Shooting	*Pressing Space*	*A bullet is created at x + 1 of the player and travels right*	*A bullet is created at x + 1 of the player and travels right*	*Yes*
Boundary (player)	*Move player to x=0 and press A*	*Player stays at x=0*	*Player stays at x=0*	*Yes*
Boundary (asteroid, bullet)	*Let bullet and asteroid leave the screen*	*Bullet or asteroid erased*	*Bullet or asteroid erased*	*Yes*
Asteroid hit player	*Let asteroid hit player*	*Asteroid destroyed and life -1*	*Asteroid destroyed and life -1*	*Yes*
Close game	*Press Q*	*Game stops running*	*Game stops running*	*Yes*
Bullet hit asteroid	*Press space and make a bullet hit asteroid*	*Both destroyed and +10 score*	*In some cases bullet passes through asteroid, but doesn't occur too often*	*No*
High score save/load	*End with score of 30*	*highscore.txt displays with 30 0 0 0 0* *When run game high score reads 30*	*highscore.txt displays with 30 0 0 0 0* *When run game high score reads 30*	*Yes*

Diagrams/screenshots

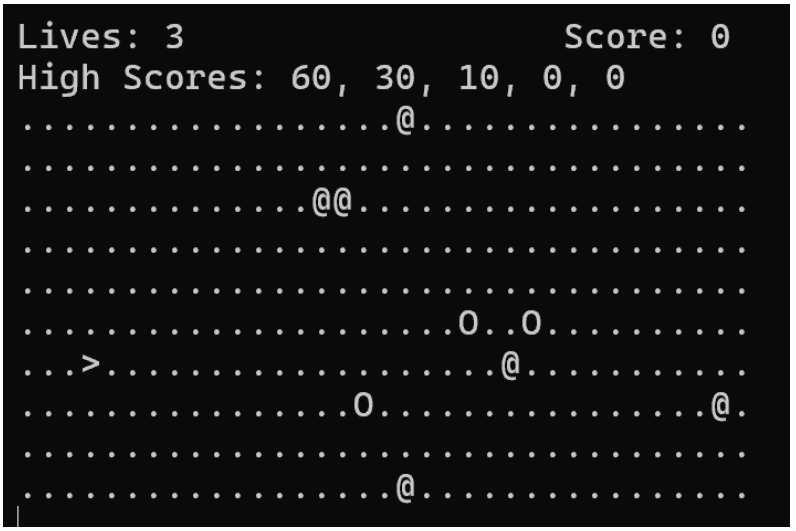
Initial ASCII diagram when generating the idea

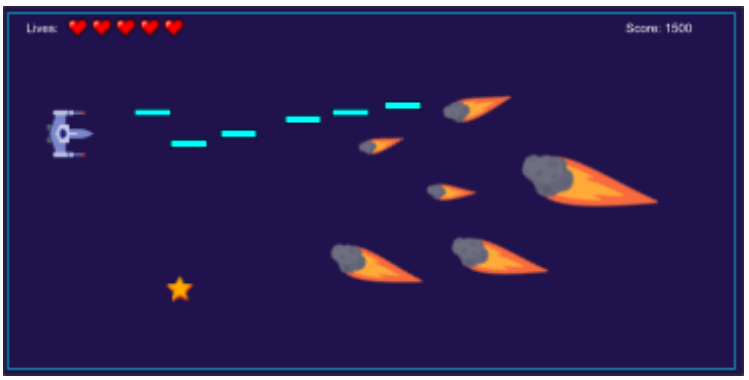


Key

- *> - Spaceship (player)*
- *O - Bullets (shot by player)*
- *P - PowerUp*
- *@ - Asteroids*
- *. - Part of stage*

Screenshot of the game in its current state





Vision for game

Game mechanics and implementation

- *Movement: player movement is controlled with standard WASD controls where each input updates player's x/y position by 1(1.0f) and limited to constraints of the map*

- *Shooting: pressing SpaceBar creates a bullet in front of the player that moves right by x += 1, the player can only do this after the 500ms cooldown has elapsed*

- *Asteroid spawning/movement: every frame there's a 10% chance of an asteroid spawning at the right edge and moves left towards the player*

```
//spawn asteroids randomly
if (rand() % 10 == 0) { //random chance to spawn an asteroid, increased frequency
    int startY = rand() % SCREEN_HEIGHT; //random y position
    asteroids.emplace_back(SCREEN_WIDTH - 1, startY); //spawn at right edge
```

- *Collisions:nested iterator loop checks collision match*
  - *Bullet:asteroid collisions = erase both and add 10 score*
  - *Player:asteroid = minus 1 life and erase asteroid*

- *Lives: levels decrease on asteroid collision but when lives reach 0 the game ends*

```
if (player.lives <= 0) {
    running = false; //endg game at 0 lives
}
```

- *Highscores:*
  - *Reads highscores from 'highscores.txt' at the start of the game and loads them onto display for player to view*

```
void loadHighScores(std::vector<int>& highScores) {         //loads high scores from file
    std::ifstream inFile("highscores.txt");
    highScores.assign(5, 0);          //reset to 0 in case file is missing or incomplete
    if (inFile.is_open()) {
        for (int i = 0; i < 5 && inFile >> highScores[i]; ++i) {}
        inFile.close();
    }
```

```
std::cout << "High Scores: ";
for (int i = 0; i < 5; ++i) {
    std::cout << highScores[i];
    if (i < 4) std::cout << ", ";
}
```

  - *Saves new high scores to 'highscores.txt' whenever player beats their previous 5 highest scores*

```
void saveHighScores(const std::vector<int>& highScores) {       //saves high scores to file
    std::ofstream outFile("highscores.txt");
    if (outFile.is_open()) {
        for (int i = 0; i < 5; ++i) {
            outFile << highScores[i];
            if (i < 4) outFile << " ";
        }
        outFile.close();
    }
```

```
if (score > *std::min_element(highScores.begin(), highScores.end())) {
    if (std::find(highScores.begin(), highScores.end(), score) == highScores.end()) {
        highScores.push_back(score);
        std::sort(highScores.begin(), highScores.end(), std::greater<int>());
        highScores.resize(5);
        saveHighScores(highScores); //saves updated high scores
    }
```

The prototype is a single file C++ console app without C primitives. The game loop runs at 6.67 FPS. (Sleep(150) to reduce flashing from cls).

Evaluation

All in all Asteroid Astronaut, is an interactive game that tests player's reactions and creates a small challenge that players can spend some time on to test their skills. The game is intended for short term use remaining simple and easy to understand by implementing widely understood controls (WASD) and mechanics (cooldowns, highscores) but not be too hard to put down to avoid addictive design.

Despite the deliverable fitting part of my vision there's a lot I wish was implemented. The game does have a few bugs, nothing too groundbreaking but It would be better if i could've ironed these out before submission. Another is the graphics, implementation of SDL for Asteroid Astronaut would really enhance the submission and would make the game a lot more immersive, unfortunately due to poor time management i wasn't able to get this implemented.

In conclusion I believe the game is a solid prototype that offers some short term enjoyment for players and even though it didn't meet my true vision the game still has something to offer.

Acceptable levels of AI use:
The table below provides the acceptable use categories for GenAI. Each assessment element may allow different uses. Please check the brief for each element carefully to see what uses are allowed.

Solo Work	**S1 - Generative AI tools have not been used for this assessment.**
Assisted Work	**A1 – Idea Generation and Problem Exploration** Used to generate project ideas, explore different approaches to solving a problem, or suggest features for software or systems. Students must critically assess AI-generated suggestions and ensure their own intellectual contributions are central.
	A2 - Planning & Structuring Projects AI may help outline the structure of reports, documentation and projects. The final structure and implementation must be the student's own work.
	A3 – Code Architecture AI tools maybe used to help outline code architecture (e.g. suggesting class hierarchies or module breakdowns). The final code structure must be the student's own work.
	A4 – Research Assistance Used to locate and summarise relevant articles, academic papers, technical documentation, or online resources (e.g. Stack Overflow, GitHub discussions. The interpretation and integration of research into the assignment remain the student's responsibility.
	A5 - Language Refinement Used to check grammar, refine language, improve sentence structure in documentation not code. AI should be used only to provide suggestions for improvement. Students must ensure that the documentation accurately reflects the code and is technically correct.
	A6 – Code Review AI tools can be used to check comments within the code and to suggest improvements to code readability, structure or syntax. AI should be used only to provide suggestions for improvement. Students must ensure that the code accurately reflects their knowledge and is technically correct.
	A7 - Code Generation for Learning Purposes Used to generate example code snippets to understand syntax, explore alternative implementations, or learn new programming paradigms. Students must not submit AI-generated code as their own and must be able to explain how it works.
	A8 - Technical Guidance & Debugging Support AI tools can be used to explain algorithms, programming concepts, or debugging strategies. Students may also help interpret error messages or suggest possible fixes. However, students must write, test, and debug their own code independently and understand all solutions submitted.
	A9 - Testing and Validation Support AI may assist in generating test cases, validating outputs, or suggesting edge cases for software testing. Students are responsible for designing comprehensive test plans and interpreting test results.
	A10 - Data Analysis and Visualization Guidance AI tools can help suggest ways to analyse datasets or visualize results (e.g. recommending chart types or statistical methods). Students must perform the analysis themselves and understand the implications of the results.
	A11 - Other uses not listed above Please specify:
Partnered Work	**P1 - Generative AI tool usage has been used integrally for this assessment** Students can adopt approaches that are compliant with instructions in the assessment brief. Please Specify: • Creating game assets (models, sprites, font and title, art, music and sound effects). • Code assistant. • Programming testing. • Readme report crafting.