Space Invaders, an arcade game about shooting aliens,

became immensely popular all around the world in 1978.

(Urban legend says it caused a national shortage of

100-yen coins in Japan.) With its blocky, iconic aliens, it

visually defined the look of commercial video games. It

also paved the way for modern video games by

pioneering the “shoot-em-up” genre and using music as

a device to influence players’ emotions. It was the first

game that introduced a difficulty curve—it got more

challenging as you played. But this wasn’t intentional; it

was actually a glitch due to technical limitations. As you

blasted away aliens, fewer aliens on the screen meant a

lighter load on the processor and, in turn, faster

rendering. The aliens sped up, as did the 4-note

soundtrack. Video games now had the ability to feel

suspenseful and formidable, and this happy accident

laid the foundation for future gameplay.

In an attempt to appreciate one of the earliest digital shooting games, a simplified version of the game Space Invaders will be created. Your problem analysis and logic formulation will be practiced in developing this program.

**DISPLAY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Display** | **Levels appearing** | **Lives to shoot** |
| Facehugger | //(~O~)\\ | ALL | 1 |
| Chestbuster | \\<^o^>// | ALL except 2 and 5 | 2 |
| Xenomorph | (((((((O,,,o) | Even levels | 4.2 |
| Queen | )))O\*\*\*O((( | 10 | 7.3 |

**GAME PLAY**

Players have to enter their player ID to start playing. There are a total of 10 levels for this game. Each player begins with 3 lives and 0 score. The live count will not reset every level but is already until level 10.

Player has to compute how many times to shoot by counting the aliens on each type and multiplying to the lives it has (refer to the previous table). Every incorrect answer should deduct 1 life. Player has to guess all correct answers to gain 1 point for each level. Completing the game results in a score of 10.

|  |  |  |
| --- | --- | --- |
| **Level** | 2 (no Chestbuster) | 10 (with Queen) |
| **Alien Display** | //(~O~)\\ //(~O~)\\ //(~O~)\\ //(~O~)\\  (((((((O,,,o) (((((((O,,,o) | //(~O~)\\ //(~O~)\\ //(~O~)\\ //(~O~)\\  //(~O~)\\ \\<^o^>// \\<^o^>// (((((((O,,,o)  (((((((O,,,o) )))O\*\*\*O((( )))O\*\*\*O(((  )))O\*\*\*O((( |
| **Computation** | Facehugger: 4 \* 1  Xenomorph: 2 \* 4.2 | Facehugger: 5 \* 1  Chestbuster: 2 \* 2  Xenomorph: 2 \* 4.2  Queen: 3 \* 7. 3 |
| **Correct answer** | Shoot Facehugger: 4  Shoot Xenomorph: 8.4 | Shoot Facehugger: 5  Shoot Chestbuster: 4  Shoot Xenomorph: 8.4  Shoot Queen: 21.9 |

**YOUR TASK**

Implement the simplified version of Space Invaders in C following the above specifications and below:

1. A welcome page displaying the Main menu (Play, High scores, Quit) is shown at the start of execution.

2. The game can only record the top 5 players’ ID and score.

3. Play menu should ask for the player ID first (consisting of 8 digits) before starting the game.

a. If player ID is already in High scores, the higher score between old and new will be recorded.

4. The number of aliens to display and shoot by the player should be randomized and have at least 1 on every level it is included (refer to the type of aliens table on the first page).

5. Displaying each character alien should use iteration, meaning printf() statements can only have one alien character e.g. printf(“(((((((O,,,o)”);

6. Design a coherent display showing all the required information separating each level. A guide on each alien type can also be shown. The screen should not be cleared in each level so it is scrollable until player loses all lives or completed all 10 levels.

7. Option to go back to the Main menu should only be displayed after each game, not showing automatically. Once chosen, the game screen will be cleared and only the Main menu is shown.

8. There should be no dead end within the whole application.

9. High scores menu will display the top 5 scores with player ID in descending order. There

should be no redundant player ID.

10. Quit option should display a closing screen and/or message before terminating the whole program

**QUESTIONS TO PONDER:**

1. What will be the input on each part of the application?

2. What will be validated for each input?

3. How can a number be randomized?

4. How can the screen be cleared after choosing to go back to the Main menu?

5. How can player ID and scores be stored and sorted?

========================================================================

**How to Approach the Machine Project**

**Step 1: Problem analysis and algorithm formulation**

Read the MP Specifications again! Identify clearly what are the required information from the user, what kind of processes are needed, and what will be the output(s) of your program. Clarify with your professor any issues that you might have regarding the machine project.

When you have all the necessary information, identify the necessary functions that you will need to modularize the project. Identify the required data of these functions and what kind of data they will return to the caller. Write your algorithm for each of these modules/functions as well as the algorithm for your main program.

**Step 2: Implementation**

In this step, you are to translate your algorithm into proper C statements. While implementing, you are to perform the other phases of program planning and design (discussed in the other steps below) together with this step.

Follow the [Linux Kernel coding standard.](https://developer.gnome.org/documentation/guidelines/programming/coding-style.html)

You may choose to type your program in a text editor or an IDE (i.e. Dev-C++ IDE) at this point. Note that you areexpected to use statements taught in class. You can explore other libraries and functions in C as long as you can clearly explain how these work. You may also use arrays, should these be applicable and you are able to properly justify and explain your implementation using these. For topics not covered, it is left to the student to read ahead, research, and explore by himself.

Note though that you are NOT ALLOWED to do the following:

● to declare and use global variables (i.e., variables declared outside any function),

● to use goto statements (i.e., to jump from code segments to code segments),

● to use the break statement to exit a block other than switch blocks,

● to use the return statement or exit statement to prematurely terminate a loop or function or program,

● to use the exit statement to prematurely terminate a loop or to terminate the function or program, and

● to call the main() function to repeat the process instead of using loops.

It is best that you perform your coding “incrementally.” This means:

● dividing the program specification into subproblems, and solving each problem separately according to your algorithm;

● coding the solutions to the subproblems one at a time. Once you’re done coding the solution for one subproblem, apply testing and debugging.

**Documentation**

While coding, you have to include internal documentation in your programs. You are expected to have the following:

● File comments or Introductory comments

● Function comments

● In-line comments

**Introductory comments** are found at the very beginning of your program before the preprocessor directives. Follow the format shown below. Note that items in between < > should be replaced with the proper information. Items in between [ ] are optional, indicate if applicable.

**Function comments** precede the function header. These are used to describe what the function does and the intentions of each parameter and what is being returned, if any. If applicable, include pre-conditions as well. Pre-conditions refer to the assumed state of the parameters.

**IMPORTANT POINTS TO REMEMBER:**

1. You are required to implement the project using the C language (C99 and NOT C++). Make sure you know how to compile and run in both the IDE (DEV-C++) and the command prompt via

gcc –Wall -std=c99 <yourMP.c> -o <yourExe.exe>

2. The implementation will require you to:

● Create and Use Functions

Note: Non-use of self-defined functions will merit a grade of 0 for the machine project.

● Appropriately use conditional statements, loops and other constructs discussed in class (Do not use brute force solution. You are not allowed to use goto label statements, exit statements. You are required to pass parameters to functions and not allowed to declare global or static variables.)

● Consistently employ coding conventions

● Include internal documentation (i.e., comments)