**SpaceXplorer**

**C Game Development Log**

**UFMFGT-15-1 Programming For Engineers 24jan\_1**

**Mini Coursework**

**Name: Yasandu Kalansooriyage**

**Student ID: 24056462**

**Table of Contents**

## Project Requirements

- The project should be a text-based game created using the C programming language where the user pilots a spaceship through a virtually created space. The basic requirements as indicated in the Coursework Brief, along with the design choices made by me are as follows:

* The minimum size of the ‘space’ should be 18 by 18 cells – 18 by 18 cells.
* There should be obstacles (asteroids or space junk) that

- only one of which can be occupied by a single cell in the world at a time,

- moves in a fixed trajectory - downwards through space

- the placement is randomized at the beginning of each game,

- can collide with the player’s spacecraft:   
 therefore the player should avoid them – only asteroids used.

* The player spacecraft

- can be moved one cell at a time in the up, down, left and right directions but not diagonally,

- should remain within the boundaries of the space,

- uses a resource which is finite and diminishing and can be replenished according to a game mechanic of the designer’s choice – fuel of spacecraft

\*- running out of this resource is a loss condition.

- should have a health monitoring system which can be checked by the player

- should be able to use/harvest the space junk in some way (designer’s choice) – the asteroids can be harvested to add to the fuel reserves of the spacecraft, and there is a cooldown time of 5 turns until the player can harvest asteroids again.

- collision with a moving asteroid is a loss condition – 3 collisions with asteroids, enemies or enemy fire is a loss condition.

- additional mechanic: A shield can be deployed by the player which consumes fuel and in return makes the player immune to all forms of collision for 2 consecutive turns.

* The game

- difficulty should be scalable which can be chosen by the user – number of asteroids that appear per row of grid increases with difficulty.

- is won when the player meets a specific threshold of the designer’s choice – survive enemy fire until the end of the level

- should read some data from a text file and write some data to a text file at some point – creating game saves as text files

- there should be a loss condition that is triggered by running out of the finite

resource mentioned before and due to the collision with an asteroid and maybe by some other factors like enemies.

- there are 4 levels: a tutorial level and levels 1 to 3.

* Enemies

- have a fixed trajectory and pattern of attack through all levels: the player must identify these patterns to have a better chance at winning the game.

- The programming requirements include **appropriate use** of **Structures, Pointers**, and **custom header files.**

## Decomposition

- Based on the design criteria listed above, the game function will be decomposed into modules as follows:

**1) World Module:**

- Manages the 18 x 18 grid

- Places asteroids per each row of grid

- Moves asteroids down the grid per each turn while generating a new row of asteroids.

- Clears and resets the gird when a new game / level / difficulty change is attempted.

- Empty grid cells = ‘.’ Asteroids = ‘@’

***Included Functions:***

* createGrid(): set up the empty 18x18 grid.
* createAsteroids(): randomly generates asteroids on the topmost row of the grid: the number of asteroids that appears in one line depends on a predetermined percentage probability setting.
* moveAsteroids(): moves all generated asteroids down the grid.
* printGrid(): prints the grid to screen

**2) Spaceship Module:**

- Icon = ‘A’ Shield Icon = ‘O’

- Handles spaceship movement (W, A, S, D for up, down, left and right)

- Manages finite fuel resources (diminishes per turn, used up by shield function, harvested by asteroids)

- Handles spaceship health (reduce health upon any form of collision)

- Implement shield function (consume fuel to activate shield for 2 turns)

- Implement asteroid harvesting (if asteroid is in one cell proximity to spaceship, command world module to remove that asteroid from the grid and increase fuel reserves by a set amount, then go into a 5 turn cooldown)

***Included Functions:***

* movement(): Handle spaceship movement based on user input.
* harvest(): Harvest nearby asteroids to refuel spaceship
* shield(): Checks fuel reserves to see if it is sufficient to deploy shield, if it is, consume fuel to activate shield for 2 turns.
* health(): Three points total, reduce health by a point if collided, display health to player.

**3) Collision Module:**

- Before printing grid for next turn, check whether the address of the destination cell of the grid already contains an obstacle (asteroid, enemy or enemy projectile), and if it does, register a collision and indicate spaceship module to reduce one health point.

- Avoid collisions when shield function is active.

***Included Functions:***

* checkCollision(): Check if spaceship’s current position aligns with an asteroid.
* handleCollision(): Handle consequences of a collision (health reduction or game over).

**4) Enemy Module:**

- Icon = ‘Y’

- Trigger enemy fire, icon = ‘+’

***Included Functions:***

* fire(): Handle enemy firing projectiles (rate of fire and directions)

**5) Main Module:**

- Control overall game flow (game start, win/loss condition monitoring, difficulty level adjustment)

- Display information essential for the player (eg: returned values from spaceship module of Health and Fuel)

- Handle enemy motion per each level based on predetermined instructions

- Handle each turn of game

- Initialize all levels and their structure

***Included Functions:***

* initialize(): Set up game and required variables based on user input for difficulty etc.
* tutorial(): Begin tutorial level
* level1() to level3(): Implementing predetermined structured levels if previous level has been won
* checkWin(): Consider all win conditions to check if player has won the level
* checkLoss(): Consider all loss conditions to check if player has lost the level.

**6) File Handling Module:**

- Save game progress and load saved game data.

***Included Functions:***

* saveGame(): Save current state of game (level, spaceship position, health, fuel etc.)
* loadGame(): Load game state from a previous save text file.

## Preliminary Design

## Detailed Design and Code

**1) World Module**

- This is where I plan to:

* + - * Create the 18 by 18 grid
      * Store addresses for the edges and corners of the grid
      * Generate asteroids
      * Move the asteroids.

- The basis for the world is to interact using pointers – the world, spaceship, asteroids, enemies and enemy projectile will all be a part of a big structure we will call “gameObjects” which will be then manipulated by the main function to implement the actual game. Therefore, the world should be a 2D array of pointers.

- The addresses of the edges and the corners of the grid will be stored in eight different arrays/variables (four corners, four for the 16 cells of the four edges). The purpose of this is to compare the address of the spaceship and these values, and if there is a match, based on the specific array/variable from where the match was found, the user will see a special message saying that they can’t move in a specific direction(s) while the program blocks that specific direction(s) of motion.

For example: Spaceship moved to top left corner cell -> address comparison -> variable with address of top left cell is a match -> Display message “You can’t move forward or to the left from here” -> block that specific motion.

- Asteroid generation depends on two factors: game difficulty level selected by user, and probability generated by program.

If the user selects ‘Easy difficulty’ – maximum 3 asteroids per line are generated  
 ‘Medium difficulty’ - maximum 4 asteroids per line  
 ‘Hard difficulty’ - maximum 5 asteroids.

A ‘dice function’ will generate a random number between 0 and 100 – based on the number generated, a predetermined set of boundaries will determine how many asteroids will appear in a line. For example:

For Easy difficulty:  
 If random number < 20 = 1 asteroid  
 20 < random number > 50 = 2 asteroids  
 50 < random number > 100 = 3 asteroids

- Asteroids will only generate at the topmost row of the grid, then they will be copied onto the next row as a new line of asteroids will be generated on the top row. This is related to the asteroid movement function.

- The main function will set all these functions into motion by including the world.h and gameobjects.h header files in the main module. A dummy testing function should be included in main so that the turn-based asteroid motion and generation can be tested.

***Structure:***

*- Create 2D array of pointers (18 by 18) linked to gameobjects structure*

*- Define all required variables*

*- Function to clear 18 by 18 grid:*

*Incremental for loop for rows (i) < 18*

*Incremental for loop for columns (j) < 18*

*grid[i][j] = empty*

*end for*

*end for*

*- Function to store grid corner and edge addresses:*

*corner1 = grid[0][0]*

*corner2 = grid[0][17]*

*corner3 = grid[17][0]*

*corner4 = grid[17][17]*

*incremental for loop for i=1, i<18*

*edges1[0][i-1] = grid[0][i]*

*edges2[17][i-1] = grid[17][i]*

*edges3[i-1][0] = grid[i][0]*

*edges4[i-1][17] = grid[i][17]*

*end for*

*- Function to determine maximum asteroids per row:*

*Return difficulty value from main*

*If difficulty = 1*

*Maximum asteroids per row = 3*

*…similarly, else ifs, difficulty 2 = max asteroids pr 4*

*Difficulty 3 = max asteroids pr 5*

*end if*

*- Function to determine asteroids in a row based on probability:*

*Generate random number between 0 and 100*

*If Maximum asteroids per row = 3*

*If random number <= 20 return 1*

*Else if random number <= 60 return 2*

*Else if random number <= 100 return 3*

*End if*

*Else if maximum asteroids per row = 4*

*…similarly as above, rand <= 15 return 1,*

*35 return 2, 75 return 3, 100 return 4*

*Else if max apr = 5*

*…similarly, rand <=10 return 1, 30 return 2, 60 return 3,*

*85 return 4, 100 return 5*

*end if*

*- Function to add asteroids randomly to a row:*

*Return max apr from previous function*

*Generate random number between 0 and 18 as rand*

*Create asteroid linked to gameobjects structure with position: rand,0*

*Place asteroid in structure with position*

*- Function to move asteroids*

*Decremental for loop for rows (i=17) > 0*

*Incremental for loop for columns (j=0) < 18*

*If grid[i-1][j] = asteroid*

*grid[i][j] = grid[i-1][j]*

*grid[i-1][j] = empty*