A Project Report

On

**Multiplayer Dungeon Quest**

Submitted in partial fulfillment of the requirement of

Project – IV

(BIT106CO)

BACHELOR IN INFORMATION TECHNOLOGY

**Submitted to**



Purbanchal University

Biratnagar, Nepal

**Submitted by**

Bhavishek Lama (310796)

Hulas Chandra Chaudhary (310819)

Ravi Hathi (310825)

**KANTIPUR CITY COLLEGE**

Putalisadak, Kathmandu

A Project Report on

**Multiplayer Dungeon Quest**

Submitted in partial fulfillment of the requirement of   
Project – IV

(BIT106CO)

BACHELOR IN INFORMATION TECHNOLOGY

**Submitted to**

Purbanchal University

Biratnagar, Nepal

**Submitted by**

Bhavishek Lama (310796)

Hulas Chandra Chaudhary (310819)

Ravi Hathi (310825)

**Project Supervisor**  
Er. Ronak Singh Karki

**KANTIPUR CITY COLLEGE**

Putalisadak, Kathmandu

Abstract

Dungeon Quest is a 2D multiplayer action-adventure game developed in Java, designed to deliver an engaging and collaborative gaming experience. Players control characters navigating a procedurally generated 50x50 tile-based dungeon, composed of ground, walls, water, and obstacles, rendered using sprite-based graphics.

The game leverages object-oriented programming principles to implement core mechanics such as sprite animation, collision detection, and real-time input handling. Players explore the dungeon, avoid obstacles, and engage in combat with enemies using keyboard-controlled movements and attacks. The multiplayer feature allows multiple players to join the same game session, cooperating or competing within the dungeon environment. The game’s modular architecture, with distinct classes for the game panel, player entities, background management and socket programming, ensures scalability and maintainability. A thread-based game loop drives the core mechanics, updating game states and rendering graphics at a consistent 60 FPS.

Dungeon Quest blends nostalgic dungeon-crawler aesthetics with multiplayer dynamics, offering a challenging and interactive experience while serving as a practical demonstration of Java-based game development and networked gameplay.

Acknowledgement

We’d like to express our heartfelt gratitude to everyone who has encouraged us to work on this project. First and foremost, we’d like to express our gratitude to the entire team of Kantipur City College (KCC) for this chance, especially the department of science and technology, who assisted us in furthering our knowledge in this field.

And we’d like to express our special gratitude to our supervisor, Er. Ronak Singh Karki. who has consistently encouraged, inspired and provided us with a wealth of information that has been beneficial. His advice was helpful in completing this project. We could not have asked for a better supervisor, counselor, or mentor.

This project would not have been possible without the assistance of each member, who offered suggestions, shared their experiences, and provided advice throughout the project. We are grateful for this. Finally, we’d like to express our gratitude to our teachers, friends and colleagues who supported us directly and indirectly throughout this project.

Preface

Welcome to "Space Impact," a classic 2D space shooter game developed using C and the Turbo C graphics library. This project combines fundamental programming concepts with creative design to deliver an engaging gaming experience reminiscent of classic arcade games.

"Space Impact" features intuitive controls, allowing players to move a spaceship, dodge enemy fire, shoot down foes while aiming for high scores. The game includes a leaderboard system for tracking top scores and offers both standard and challenge modes for varying difficulty levels.

Creating this game has been an educational journey, involving memory management, data structures, real-time input handling, and graphics rendering. It serves as both an entertaining game and a learning tool, demonstrating the application of basic programming principles in game development.

I hope you enjoy playing "Space Impact" and find it both fun and insightful.

Declaration

I declare that this semesters project report titled “Multiplayer Dungeon Quest” submitted in partial fulfillment of the Bachelor of Information Technology, is a record of original work carried out by me under the supervision of Er. Ronak Singh Karki. This work has not been submitted for the award of any other degree at this or any other institution or university. In keeping with the ethical practice of reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Bhavishek Lama

Hulas Chandra Chaudhary

Ravi Hathi

Supervisor’s Approval

This is to certify that this semester the project entitled “Multiplayer Dungeon Quest” undertaken and demonstrated by Bhavishek Lama, Ravi Hathi and Hulas Chandra Chaudhari has been successfully completed under my supervision as a partial fulfillment of the requirements of the degree of BIT 4th semester under Purbanchal University. I, henceforth, approve this project to be awarded the certificate by the concerned authority.

During supervision, I found students hardworking, skilled and ready to undertake any professional work related to this field in the future.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Er. Ronak Singh Karki

Project Supervisor

Department of IT

Kantipur City College

**Certificate from Department**

Following the Supervisor's Approval and Examiners' Acceptance, the project entitled “Multiplayer Dungeon Quest” submitted by Bhavishek Lama, Ravi Hathi and Hulas Chandra Chaudhari as a partial fulfillment of the requirements for the degree of BIT 4th under Purbanchal University, has been officially awarded by this certificate. I wish the students all the best for their future endeavors.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Saroj Pandey

HoD

Department of IT

Kantipur City College

**Table of Contents**

**Table of Contents**

[1 Introduction 1](#_Toc207388470)

[1.1 Overview 1](#_Toc207388471)

[1.2 Problem Statement 1](#_Toc207388472)

[1.3 Objectives 2](#_Toc207388473)

[1.4 Features 2](#_Toc207388474)

[1.5 Significance 2](#_Toc207388475)

[1.6 Scope and Limitations 3](#_Toc207388476)

[1.7 Documentation Organization 3](#_Toc207388477)

[2 Literature Review 4](#_Toc207388478)

[2.1 Overview of Existing System 1 (Rogue) 4](#_Toc207388479)

[2.2 Overview of Existing System 2 (Gauntlet) 4](#_Toc207388480)

[2.3 Analysis of Literature Review 6](#_Toc207388481)

[3 Methodology 7](#_Toc207388482)

[3.4 Roles and Responsibilities 8](#_Toc207388483)

[4 System Analysis 9](#_Toc207388484)

[4.1 Program Components 9](#_Toc207388485)

[4.2 Requirement Analysis 9](#_Toc207388486)

[4.2.1 Requirement Gathering 9](#_Toc207388487)

[4.2.2 Functional Requirement 10](#_Toc207388488)

[5 System Design and Implementation 11](#_Toc207388489)

[5.1 Gantt Chart 11](#_Toc207388490)

[5.2 Algorithm 12](#_Toc207388491)

[5.3 Flowcharts 18](#_Toc207388492)

[6 Testing and Debugging 25](#_Toc207388493)

[6.1 Tools used in Testing 25](#_Toc207388494)

[6.2 Test case of 26](#_Toc207388495)

[7 Conclusion 28](#_Toc207388496)

[References 29](#_Toc207388497)

# 1 Introduction

Space Impact is a 2D space shooter game designed to deliver an engaging and challenging experience to players. Developed using the C programming language, this game uses the graphics.h library for graphical rendering and the conio.h library for input handling. The project not only provides entertainment but also serves as a practical example of various fundamental game development concepts, including data structures, dynamic memory allocation, and collision detection.

## 1.1 Overview

The game starts with the main menu, offering options to start the game, view the high score and exit the game. The player gets five life by default.

The player controls a spaceship (symbol ‘A’) using the keyboard keys 'a', 'w', 's', 'd' to move ‘q’ to quit ‘e’ to quit and save game ‘p’ to pause the game and spacebar to shoot bullets. The enemy ships (symbol 'V') move down the screen and shoot bullets (symbol '|') towards the player. The player must avoid collisions with enemy ships and enemy bullets. Each collision reduces the player's life. Players can also shoot bullets to destroy enemies, earning points for each successful hit. The game will be developed using C language without any external libraries.

## 1.2 Problem Statement

The main objective of any game is to provide a gaming experience where players don’t get bored easily and have better game and player interaction. Space impact is a 2D space shooter game, designed to provide a captivating and challenging experience for players but it lacks the engaging factor, not only that the enemy battleships follow a pattern where they would spawn in a similar manner and the obstacles pattern were also same and not only that it only has one mode which leads to players getting bored and leaving the game for other more engaging games.

## 1.3 Objectives

The objectives of the proposed project are as follows:

* To develop the user-friendly interface console game
* To implement the use of control statements, loops, file handling concept by using C programming.

## 1.4 Features

The features of the space impact are as follows:

* Score Tracking System: This game will keep the record so that the players can view score during the game and creating the new high scores.
* Leaderboard System: Maintain a leaderboard to track and display the top 5 high scores.
* Standard Mode: Play with multiple lives, allowing a more forgiving experience.
* Challenge Mode: Play with only one life, offering a higher difficulty level for advanced players.
* Pause and Resume: Pause the game at any time and resume from where you left off.
* Quit and save: Player can save their game score even if they are quitting the game prematurely

## 1.5 Significance

Space Impact is not just a fun game, it's a great way to learn basic programming skills like managing memory, handling input, and creating 2D graphics. It brings back the excitement of classic arcade games while challenging you to improve your reflexes and strategic thinking. Compete for high scores on the leaderboard, share your experiences with others, and even modify the game, it's a project that combines entertainment with learning. Whether you're reliving the nostalgia of old-school gaming or diving into game development, Space Impact offers something for everyone to enjoy and explore.

# 1.6 Scope and Limitations

**Scope:**

The scope of Space Impact as an educational tool is it serves as a practical platform for learning fundamental programming concepts. Its gameplay offers both casual entertainment and a challenge to improve reflexes and strategic thinking. The game helps community engagement through shared resources and collaborative enhancements, nurturing a culture of creativity and innovation within game development circles. Space Impact also contributes to the preservation and appreciation of classic arcade gaming experiences, ensuring its enduring enjoyment and relevance across different generations of gamers and developers.

**Limitations:**

While developing the program we thought of adding some more features but couldn’t due to our inefficiency

* Single user
* Limited user interface
* Leveling mode

## 1.7 Documentation Organization

This documentation is a comprehensive report on the project to create a Space Impact. It also includes all background information regarding the project under Chapter 1 so that even a new reader can understand what the project is about. Similarly, it also clarifies how the program works and its main features and objectives. Alongside the functions of the program, it also shows how this program is different from the other programs present on internet. It includes how the program is used in real life. Similarly, it includes the detailed information on how the program was made.

# 2 Literature Review

We played and reviewed many similar games to familiarize ourselves with the dungeon crawler we are trying to make. Among all the games, two of them were particularly similar to our project, but as expected, they shared similarities in graphics and features while offering unique elements for inspiration.

## 2.1 Overview of Existing System 1 (Shattered Pixel Dungeon)

Shattered Pixel Dungeon is a traditional roguelike dungeon crawler RPG, available on platforms like Android, iOS, and Steam. Players control a hero descending through procedurally generated dungeon levels, fighting enemies, collecting items, and managing resources in a turn-based grid system with pixel art graphics. The game emphasizes replay ability through randomization, class choices (warrior, mage, rogue, huntress).

**Pros:**

* Infinite replay ability through procedural generation
* Strategic depth and mechanics
* Simple yet addictive gameplay loop

**Cons:**

* Steep learning curve for new players
* No multiplayer support
* No progression saves
* frequent deaths

## 2.2 Overview of Existing System 2 (Elona+)

Elona+ offering an open-world sandbox experience with roguelike elements. Players create a character from various races and classes, exploring a vast overworld, completing quests, breeding monsters, and delving into dungeons in a turn-based system with tile-based graphics. It features deep simulation mechanics.

**Pros:**

* Better graphics
* Highly customizable open-world
* Forgiving mechanics compared to traditional roguelikes

**Cons:**

* Overwhelming and confusing for beginners due to complexity
* Single-Player
* Limited QoL features

## 2.3 Analysis of Literature Review

|  |  |  |  |
| --- | --- | --- | --- |
| **Features** | **Shattered Pixel Dungeon** | **Elona+** | **Our game** |
| **Open-world** | Yes | Yes | Yes |
| **Single player** | Yes | Yes | Yes |
| **Forgiving mechanics** | No | Yes | No |
| **Multiplayer** | No | No | Yes |
| **Progressive saves** | No | Yes | Yes |
| **Game sound** | Yes | yes | Yes |

# 3 Methodology

The methodology for developing the space impact game in C involves the modified waterfall method which involves structured planning, iterative development, and input. It starts with defining requirements and detailed design, followed by implementation and continuous testing and maintenance.

**Requirements:**

The game includes continuous addition and removal of the game objects like bullet and enemies. And the game logic to determine the object collisions and then the removal of the collided objects, decreasing the life, saving the score in .txt file.

**System Design:**

An algorithm and flowchart were designed and the timeline chart for the development of the game was prepared.

**Implementation:**

We developed the game based on approved design.

**Testing and Debugging:**

Testing and debugging of the programming was done in every phases as to not face a bigger problem later on.

**Deployment and Maintenance:**

Continuous maintenance of the system was done to run the program smoothly.

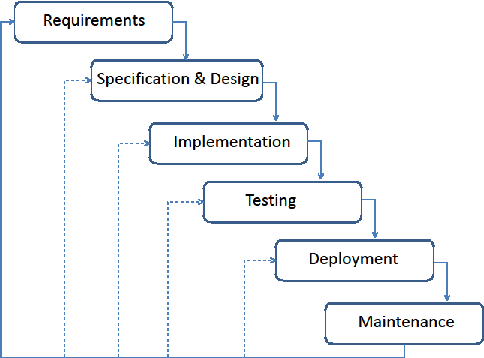


Fig: Modified Waterfall Model

## 3.4 Roles and Responsibilities

|  |  |
| --- | --- |
| **Team Member** | **Task Performed** |
| Bhavishek Lama | * Coding * Documentation |
| Rinkesh Chaudhari | * Coding |
| Sahil Karki | * coding |

# 4 System Analysis

## 4.1 Program Components

|  |  |
| --- | --- |
| **Preprocessors** | **Characteristics** |
| #include<stdio.h> | All standard library functions for file input and output are included in it. |
| #include<stdlib.h> | It is used for memory allocation and process control. |
| #include<time.h> | It allows us to use time and clock functions. |
| #include<graphics.h> | It allows us to use a graphic function. |
| #include<conio.h> | Provides functions like getch, khat, closer for console-based input and output operations. |
| #include<string.h> | It allows us to use string functions. |

## 4.2 Requirement Analysis

It involves identifying and documenting the functional and non-functional requirements that define the project’s scope and objective.

### 4.2.1 Requirement Gathering

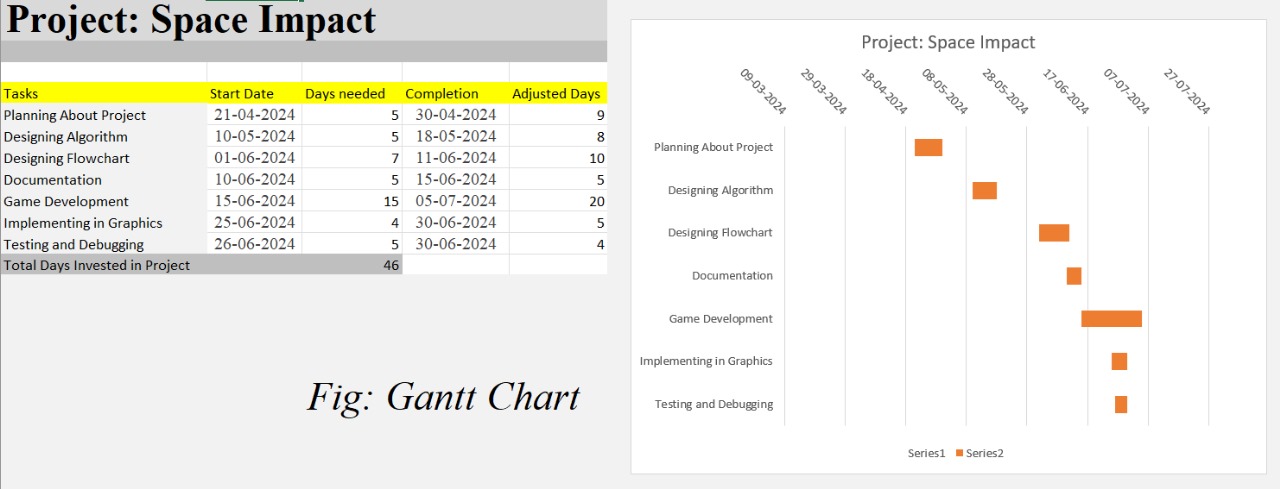
|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Menu | To give more option to player |
| modes | To make the game more competitive |
| Leaderboard | To keep the record of the best players |
| High score | To display a highest score while playing |
| Name saving | To show the name of high score holders |

### 4.2.2 Functional Requirement

|  |  |
| --- | --- |
| **Function name** | **Description** |
| create Object () | Function to create game objects |
| add Object () | Function to add the game objects |
| remove Object () | Function to remove the game objects |
| delete game () | Function to delete all the allocated memory |
| put game () | Function to create spaceships |
| get Input () | Function to check keyboard input and return the input |
| game logic (); | Function to update game |
| delay () | Function to delay\pause the execution |
| render () | Function to draw the game characters in the graphical windows |
| move () | Function to control the movement, shooting during the runtime |
| game () | Function to check game during runtime like pause, quit and whether the game has ended or not |
| sort Leaderboard () | Function to short the leaderboard based on the scores |
| save Leaderboard () | Function to write the scores in the leaderboard |
| load Leaderboard () | Function to read the scores in the leaderboard |
| addScoreToLeaderboard() | Function to insert new high score in the .txt file |
| rendermenu () | Function to display menu and take the player to the chosen option |
| renderMenu () | Function to display the menu in graphic mode |
| getHighestScore () | Function to get the highest score from the leaderboard |

# 5 System Design and Implementation

## 5.1 Gantt Chart



## 5.2 Algorithm

**Algorithm of the program**

**STEP 1:** Start

**STEP 2:** Define constants for game parameters

**STEP 3:** Create a structure which contains the game object properties

**STEP 4:** Create a linked list structure

**STEP 5:** Create a function to create node where the memory is allocated and the property of the game object is provided

**STEP 6:** Create a function to remove node from the list, it consists of a loop which will iterates till it finds the link pointing towards the object to be removed then a new node is pointed by the link using temporary node then the node to be removed will be removed using free function

**STEP 7:** Create a function to add node in the list, it consists of a loop which will iterates till it finds the link pointing towards the NULL, then it is set to point at the new node using the arrow operator

**STEP 8:** Create a function to give property to the game objects like an enemy spaceship

**STEP 9:** Create a function to take input using khat and getch function.

**STEP 10:** Create a function to move and shoot, where the input taken from step no. 8 will be used to move up, down, left, right and shoot

**STEP 11:** Create a function to update the game objects like movement of our bullet, enemy, enemy bullet, obstacle and to check the collision

**STEP 12:** Create a function to render the game objects, score, high score, life and finally the gave over message and add a delay function to show the message for longer duration

**STEP 13:** Create a function to open the file containing the high score

**STEP 14:** Create a function to save the score in the file

**STEP 15:** Create a function to delete the allocated memory to the game object

**STEP 16:** Create a function to run the game where graphic mode is opened and all essential functions are called to run the game

**STEP 17:** Create a function to render the high score

**STEP 18:** Create a function to play a multiplayer mode using time where the step 16 function will run for certain amount of time then after that another player will take over to get high score

**STEP 19:** Create a main function to where a do while loop is used to run the game in menu-based system

1. Press 1 to play survival
2. Press 2 to view high score
3. Press 3 to play time mode
4. Press 4 to exit Enter your option

**STEP 20:** if 1 then call step no. 16 function

**STEP 21:** if 2 then call step no. 17 function

**STEP 22:** if 3 then call step no. 18 function

**STEP 23:** if 4 then print exiting.... then break the loop

**STEP 24:** the loop will iterate till the player exits or press q to quit the game

**Algorithm for create Object**

1. Allocate memory for a new Game Object.
2. Set the object's x position.
3. Set the object's y position.
4. Set the object's symbol.
5. Initialize the object's next pointer to NULL.
6. Return the newly created object.

**Algorithm for add Object**

1. If the list is empty:

* Set the list's head to the new object.

1. else:

* Traverse the list to find the last object.
* Set the last object's next pointer to the new object.

**Algorithm for remove Object**

1. If the object to remove is the head of the list:

* Set the list's head to the next object.

1. else:

* Traverse the list to find the object just before the one to remove.
* Set the next pointer of the previous object to the next pointer of the object to remove.

1. Free the memory allocated for the object to remove.

**Algorithm for delete game**

1. While the spaceship list is not empty:

* Remove the first spaceship object.

1. Repeat the same process for enemies, bullets, enemy bullets, and obstacles lists.

**Algorithm for put game**

1. Create the spaceship object at the center bottom of the screen.
2. Initialize enemies and other game objects as needed.

**Algorithm for get Input**

1. If a key is pressed:

* Return the pressed key character.

1. else:

* Return '\0'.

**Algorithm for game logic**

1. Update the positions of bullets, enemies, enemy bullets, and obstacles.
2. Handle collisions between bullets/enemies and spaceship /enemies
3. Apply game rules (e.g., score updates, life updates).

**Algorithm for delay**

1. Record the start time.
2. Loop until the current time is greater than or equal to the start time plus the delay duration.

**Algorithm for render**

1. Clear the screen.
2. Draw the spaceship.
3. Draw all enemies, bullets, enemy bullets, and obstacles.
4. Display the score and remaining lives.

**Algorithm for move**

1. Based on the input character, update the spaceship's position:

* If input is a, move spaceship left.
* If input is d, move spaceship right.
* If input is w, move spaceship up.
* If input is s, move spaceship down.

1. Ensure the spaceship does not move out of bounds.

**Algorithm for game**

1. initialize game state by calling put game.
2. Enter the main game loop:

* Get user input.
* Update game state based on input and game logic.
* Render the game objects and UI elements.
* Delay for a short period to control game speed.

1. Exit the loop if the game is over (e.g., player runs out of lives).

**Algorithm for sort Leaderboard**

1. Implement a sorting algorithm to sort the leaderboard array in descending order based on scores.

**Algorithm for save Leaderboard**

1. Open the file space.txt for writing.
2. Write each leaderboard entry (name and score) to the file.
3. Close the file.

**Algorithm for load Leaderboard**

1. Open the file space.txt for reading.
2. Read each leaderboard entry (name and score) from the file into the leaderboard array.
3. Close the file.

**Algorithm for addScoreToLeaderboard**

1. Create a new leaderboard entry with the given name and score.
2. Add the new entry to the leaderboard array.
3. Sort the leaderboard.
4. Save the updated leaderboard to the file.

**Algorithm for renderMenu**

1. Display the main menu with options (Start Game, Challenge Mode, High Scores, Exit).
2. Wait for user input to select an option.
3. Execute the selected option (e.g., start the game, show high scores, exit).

**Algorithm for getHighestScore**

1. Initialize a variable to store the highest score.
2. Traverse the leaderboard array.
3. For each entry, update the highest score if the entry's score is higher.
4. Return the highest score.

**Algorithm for main**

1. Seed the random number generator.
2. Call menu to start the game.
3. Call delete game to clean up allocated memory before exiting.
4. Return 0.

**Algorithm for main menu**

1. Display options:

* Start Game
* Leaderboard
* Exit.

1. ‘1’: Start the game. Call game(). After game over, prompt for player's name
2. '2': Display leaderboard.Show leaderboard entries .Wait for user input to return to menu
3. '3': Exit.
4. Game Setup (putgame) Create spaceship object. Create and add initial enemies. Game Loop (game):Initialize graphics.Loop until game over or user quits: Check for user in put.'p': Toggle pause .'q': Quit game ‘e’ Quit and save game. If not paused: Call move with user input to update spaceship position and shoot bullets. Call Gamelogic to update game state. Call render to draw the game.

**Algorithm for collision detection**

1. It checks weather the game objects are in same x axis and if the distance in y axis between the objects is 0
2. If the objects are our\_bullet and enemyship
3. 10 will be added in our score and the game objects are removed and new enemyship is instantly created with different x coordinate
4. If the objects are enemy\_bullet and our\_ship
5. 1 point is deducted from our life

## 5.3 Flowcharts

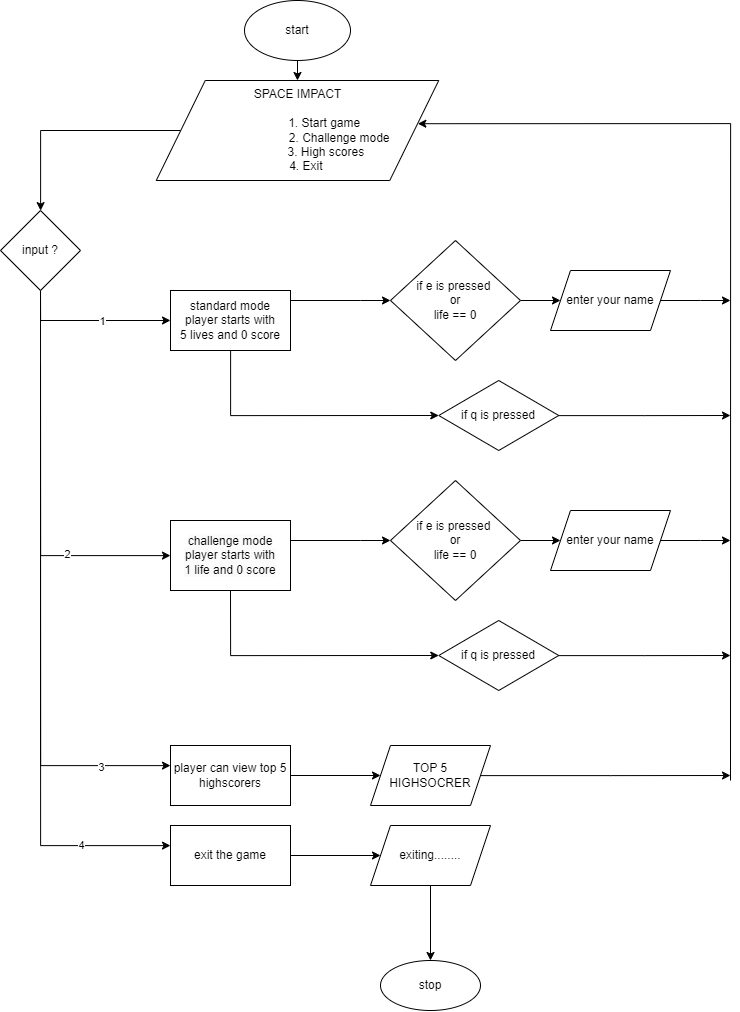
****

Figure 5.3.1: Flowchart for Game

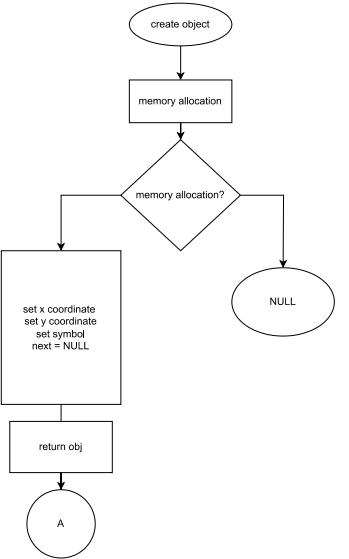
****

Figure 5.3.2 :Flowchart for createobject

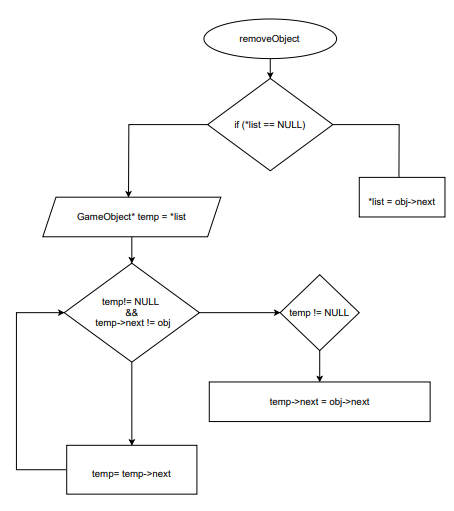
****

Figure 5.3.3: Flowchart for removeobject

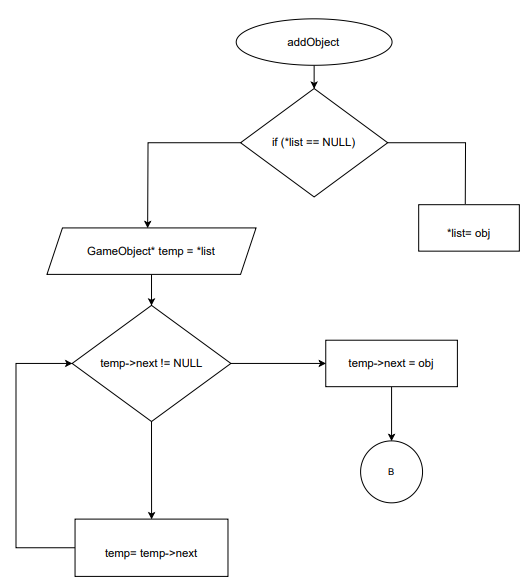
****

Figure 5.3.4: Flowchart for addobject

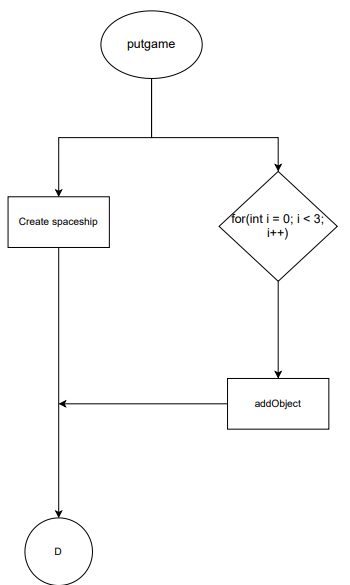
****

Figure 5.3.5: Flowchart for putgame

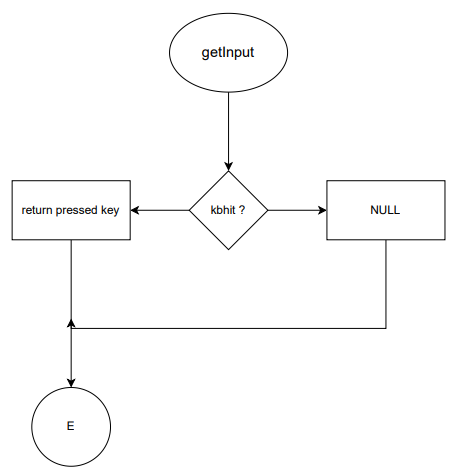
****

Figure 5.3.6: Flowchart for getinput

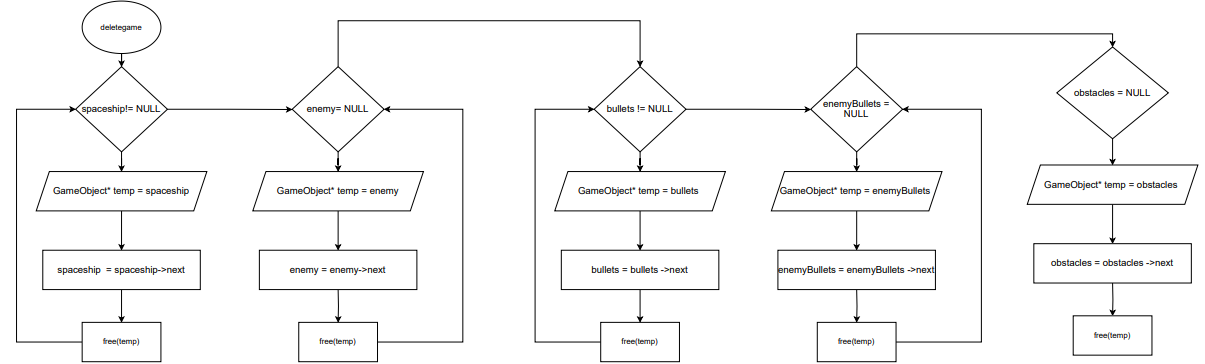


Figure 5.3.7: Flowchart for deletegame

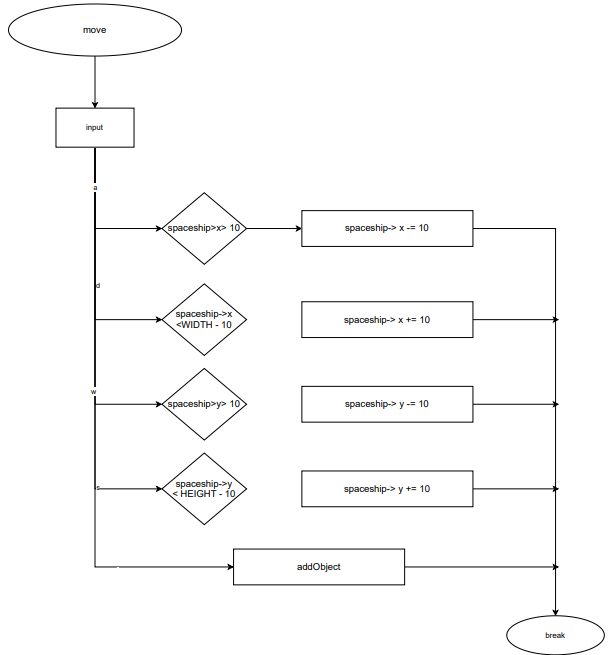
****

Figure 5.3.8: Flowchart for move

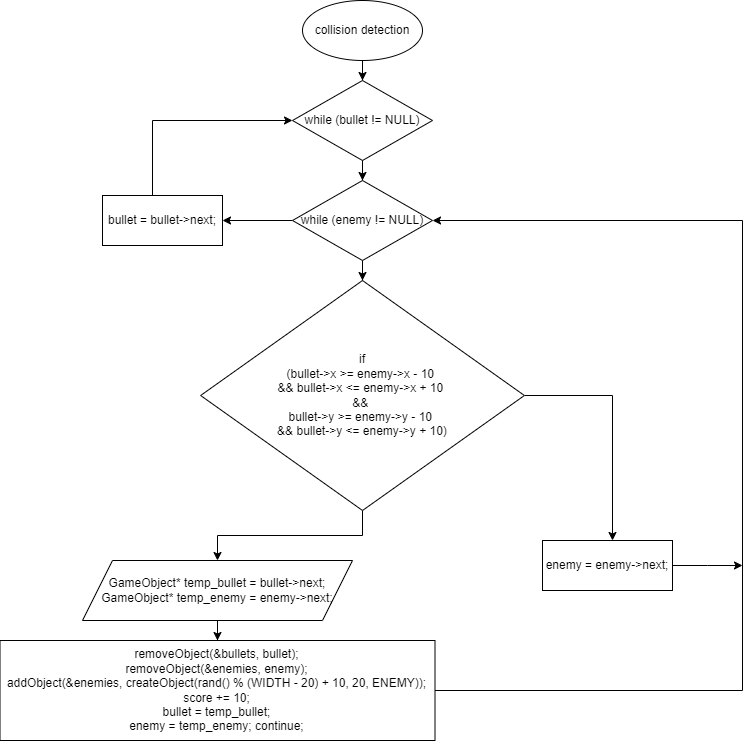


Figure 5.3.9: Flowchart for collision detection

# 6 Testing and Debugging

## 6.1 Tools used in Testing

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Tools** | **Specification** |
| 1 | Hardware (Laptop) |  |
| 2 | Testing software | Turbo c++ |

## 6.2 Test case of

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SN** | **Test objective** | **Expected Output** | **Result** | **Remarks** |
| **1** | Creation, addition and removal of nodes in linked list | To create, add and remove nodes in linked list | error | failure |
| **2** | Creation, addition and removal of nodes in linked list  (after debugging) | To create, add and remove nodes in linked list | Successful after declaring a head pointer | Pass |
| **3** | Collision detection | To detect collision between two objects | Detected the collision | Pass |
| **4** | Boder detection | To remove the enemy after crossing border | Successfully detected and removed | pass |
| **5** | Displaying the game objects | To display the game ojects  (‘A’ ‘V’) | nothing | failure |
| **6** | Displaying the game objects  (after debugging) | To display the game ojects  (‘A’ ‘V’) | Displayed the objects after using settextstyle ) | Pass |
| **7** | Move function | To move spaceship | Successful after changing the value of x | Pass |
| **8** | To check the function which pauses the game | Game coming to halt | Game continued | failure |
| **9** | To check the function which pauses the game  (after debugging) | Game coming to halt | Game came to halt | pass |
| **10** | Render menu and use menu system | Render menu in graphics mode | rendered | pass |
| **11** | Leaderboard system | To save load and add score in a file | Saved the score | pass |
| **12** | Sorting leaderboard | To display leaderboard in descending order | Displayed in descending order | pass |
| **13** | Runnig the game | To run game successfully | Couldn’t go back to menu after quitting | failure |
| **14** | Runnig game | To run game successfully | Couldn’t play game after dying | failure |
| **15** | Running game | To run game successfully | Successful after deleting the allocated memory for the game objects | pass |

# 7 Conclusion

In conclusion, the Space Impact game, developed using C programming, offers an engaging and accessible gaming experience. Despite its limited user interface, the game provides entertainment. Further enhancements could improve user interaction so that this game could give a better experience to the user.

# References

* Balagurusamy, E. (2011). *Programming in ANSI C*. Tata McGraw Hill Education.
* Mittal, A. (2010). *Programming in C: A practical approach*. Dorling Kindersley (India)