**THE CLASH OF ARMS**



**PRESENTED BY GROUP MEMBERS:-**

AHMED HUSSAIN(CT-294)

SHAHEER(CT-276)

Mohammad Ibrahim (CT-285)

ALIYAN SHAIKH(CT-252)

**REPORT ON “CLASH OF ARMS”**

FIGHTING GAME MADE ON C LANGUAE

# Table of Contents

1. **Executive Summary** .................................................... 1
2. **Introduction** ............................................................... 1
3. **Background Study** ................................................... 2
4. **Concepts from the Course Used in the Project** ……….3
5. **Challenges in Manual Inventory Management** ............ 3
6. **Solutions** ................................................................. 3
7. **Technical Tools** ....................................................... 4
8. **Programming Language** .......................................... 4
9. **Development Process** ............................................ 5

oInitial Setup ............................................................ 5

oPlayer Mechanics .................................................... 5

oProjectile System ..................................................... 5

oHealth and Collision Detection .............................. 6

oGame Loop ............................................................... 6

1. **Data Storage** ......................................................... 6
2. **Libraries Used** ....................................................... 7
3. **Testing Tools** ....................................................... 7
4. **Features in Detail (With Code and Algorithm)** .......... 8

* Character Movement and Animation ....................... 8
* Projectile Firing ...................................................... 9
* Health Bar Rendering ............................................. 9

1. **Handling Player Input** .......................................... 10
2. **Methodology** ..........................................................
3. **Maintenance and Update** ..................................... 11
4. **Time and Durations (Project Phases)** ..................... 11
5. **Risk Factors and Challenges** ................................ 12

* Player Input Lag .................................................... 12
* Projectile Collision Detection ............................... 12

1. **Future Scope** .......................................................... 12

* Multiplayer Features .............................................. 12
* Enhanced AI ............................................................. 12
* Additional Features ................................................. 12

1. **Unsuccessful Attempt** .......................................... 12
2. **Contribution of Members** ..................................... 13
3. **Conclusion** .............................................................
4. **References** .............................................................. 13

# Clash of Arms: 2-Player Fighting Game

**1: Executive Summary:**

This document outlines the design and development of **Clash of Arms**, a 2-player fighting game implemented using SDL (Simple Direct Media Layer). The game provides a real-time combat environment where two players control characters who can move, jump, attack, and defend. The game includes key elements like health tracking, projectile combat, and dynamic animations based on player actions.

**2: Introduction:**

**Clash of Arms** is an action-packed fighting game developed with the goal of creating an interactive and engaging experience for players. The game features two characters with different animations and abilities, offering both strategic and action-driven gameplay.

The game’s mechanics allow for player-controlled movements such as walking, jumping, defending, and light/heavy attacks. It also features projectile-based combat, where players can shoot projectiles at each other while managing their health through a visual health bar.

**3: Background Study:**

The development of **Clash of Arms** utilizes SDL, a widely used library for creating games and multimedia applications. SDL is cross-platform and works well for games that involve 2D graphics, sound, and user input. In this game, SDL handles:

* **Graphics Rendering**: Loading and displaying textures for characters, backgrounds, and projectiles.
* **Event Handling**: Detecting player inputs from the keyboard (movement, jumping, attacking).
* **Game Loop**: Continuously updating the game state and rendering the scene.

This game can be seen as a starting point for more complex 2D fighting games, showcasing fundamental concepts such as collision detection, animation cycles, and projectile management.

**4: Concepts from the Course Used in the Project:**

In this project, we applied several key concepts learned during the course:

1. **Functions**: We organized and managed game actions, like player movement and projectile firing.
2. **Structures**: We used structures (e.g., Player, Projectile) to store and manage information about characters and objects.
3. **Game Loop**: The continuous game loop concept was used to keep the game running smoothly, updating and rendering actions.
4. **Collision Detection**: We applied the logic of checking when projectiles hit players to decrease their health.
5. **Memory Management**: Using C’s low-level memory control helped manage game resources and performance efficiently.

**5: Challenges in Manual Inventory Management:**

* **State Tracking**: The game needs to track the state of the player at any given moment (e.g., jumping, walking, attacking). Implementing this promptly with proper visual feedback for the player can be challenging.
* **Projectile Management**: Handling multiple projectiles on-screen, determining if they hit a player, and applying the correct effects like health reduction is complex.
* **Health Bar Display**: Creating an intuitive health bar that reacts to player damage and reflects the player’s current health is essential for gameplay clarity.

**6: Solutions:**

* **State Management**: The player’s actions are managed by flags within the Player struct (isJumping, isAttackingLight, etc.). These flags control animations and game logic (e.g., preventing attacks during a jump).
* **Projectile Handling**: The projectiles are created dynamically and controlled via the Projectile struct. Each projectile has a velocity (vx) and damage properties. The projectile’s collision with the target (another player) is checked in each game loop.
* **Health System**: Health is visualized using a health bar that changes color based on the player's remaining health. This is updated in real-time and provides crucial feedback to the players.

**7: Technical Tools:**

* **SDL (Simple DirectMedia Layer)**: Primary library used for creating the game’s graphics, handling input, and managing the game loop.
* **SDL\_image**: Used for loading images and textures into the game.
* **C Programming Language**: The game is implemented using C, chosen for its efficiency and control over memory management and performance.

**8: Programming Language:**

The game is written in C, a powerful systems programming language known for its speed and efficiency. C allows precise control over memory management and is wellsuited for game development.

**9: Development Process:**

The development process involved several stages:

1. **Initial Setup**: Setting up SDL and preparing the basic window and renderer.
2. **Player Mechanics**: Implementing player movement, jumping, attacking, and animation.
3. **Projectile System**: Creating projectiles that interact with players and damage them upon collision.
4. **Health and Collision Detection**: Implementing health reduction and collision checks between projectiles and players.
5. **Game Loop**: Ensuring that the game runs in a continuous loop, updating the game state and rendering it on the screen.

**10: Data Storage:**

The game doesn’t require complex databases or external storage. All player data (such as health, position, velocity, etc.) are stored in memory in the form of structures (Player and Projectile).

**11: Libraries Used:**

* **SDL2**: The core library for game rendering and input handling.
* **SDL\_image**: A supplementary library for loading images in various formats.

**12: Testing Tools:**

* **SDL Debugging**: The SDL library comes with various debugging tools to help visualize rendering, track resource usage, and identify bugs.
* **Manual Playtesting**: Testing the game by playing through it multiple times, ensuring that the mechanics (such as projectile collision and health bar behavior) work correctly.

**13: Features in Detail (With Code and Algorithm):**

**Character Movement and Animation:**

Player movement is handled by checking for key presses (e.g., left arrow key for moving left, right arrow key for moving right):

if (keystate[SDL\_GetScancodeFromKey(leftKey)]) { player->vx = -2; // Move left player-

>facingRight = 0; // Player faces left

}

else if (keystate[SDL\_GetScancodeFromKey(rightKey)]) { player->vx = 2; // Move right player->facingRight = 1;

// Player faces right }

**14: Projectile Firing:**

When the player presses the attack button, a projectile is created in the direction the player is facing and given velocity based on player position:

void ThrowProjectile(Projectile\* projectile, Player\* player, SDL\_Texture\* projectileTexture, int damage) {

if (!projectile->isActive && player->isAttackingLight) {

projectile->x = player->x + (player->facingRight ? 50 : -50); // Fire direction

projectile->y = player->y + 50;

projectile->vx = player->facingRight ? 10 : -10; // Horizontal velocity projectile-

>isActive = 1; projectile->texture = projectileTexture; projectile-

>damage = damage;

player->isAttackingLight = 0; // Reset attack state

} }

**15: Health Bar Rendering:**

The health bar changes color and size as the player’s health decreases, using the RenderHealthBar function:

void RenderHealthBar(Player\* player, int x, int y) { int healthWidth = (player->health \* 200) / 100; // Health width relative to max health

SDL\_SetRenderDrawColor(renderer, 50, 50, 50, 255); // Dark gray background

SDL\_Rect healthBar = { x, y, healthWidth, 20 };

SDL\_RenderFillRect(renderer, &healthBar);

}

**16: Loading Textures (Images)**

Textures are the images used for characters, backgrounds, and other elements of the game. We load these images using the LoadTexture() function:

SDL\_Texture\* LoadTexture(const char\* file) {

SDL\_Surface\* tempSurface = IMG\_Load(file);

SDL\_Texture\* texture = SDL\_CreateTextureFromSurface(renderer, tempSurface); SDL\_FreeSurface(tempSurface);

return texture;

}

**17: Player Structure and Animation States**

We need a way to represent each player in the game. This is done using a Player structure that holds the player’s position, health, movement, and animation states.

typedef struct { float x, y; float vx, vy; int health; int isJumping; int isAttackingLight; int isAttackingHeavy; int isDefending; int facingRight; int walkFrame; int idleFrame; int jumpFrame; int lightAttackFrame; int heavyAttackFrame; Uint32 lastAnimTime;

} Player;

**18: Main Game Loop**

The game runs in a loop that constantly checks for input, updates the game world, and redraws everything. Here’s the core of the game loop:

void GameLoop() {

Player player1 = { 50, 500, 0, 0, 100, 0, 0, 0, 1, 0, 0, 0, 0, 0 }; Player player2 = { 990, 500, 0, 0, 100, 0, 0, 0, 1, 0, 0, 0, 0, 0 }; int quit = 0;

SDL\_Event event;

while (!quit) { while (SDL\_PollEvent(&event)) { if (event.type == SDL\_QUIT) quit = 1;

}

HandlePlayerMovement(&player1, SDLK\_a, SDLK\_d, SDLK\_w, SDLK\_q, SDLK\_s);

HandlePlayerMovement(&player2, SDLK\_LEFT, SDLK\_RIGHT, SDLK\_UP,

SDLK\_RCTRL, SDLK\_DOWN);

UpdatePlayerAnimation(&player1);

UpdatePlayerAnimation(&player2);

ApplyGravity(&player1);

ApplyGravity(&player2);

ThrowProjectile(&player1Projectile, &player1, lightProjectileTexture, 10); ThrowProjectile(&player2Projectile, &player2, lightProjectileTexture, 10);

UpdateProjectile(&player1Projectile, &player2); UpdateProjectile(&player2Projectile, &player1); if (player1.health <= 0 || player2.health <= 0) quit = 1;

SDL\_RenderClear(renderer);

RenderTexture(backgroundTexture, 0, 0, 1200, 800);

RenderPlayer(&player1);

RenderPlayer(&player2); if (player1Projectile.isActive)

RenderTexture(player1Projectile.texture, player1Projectile.x, player1Projectile.y,

40, 40); if (player2Projectile.isActive)

RenderTexture(player2Projectile.texture, player2Projectile.x, player2Projectile.y, 40, 40);

RenderHealthBar(&player1, 50, 20);

RenderHealthBar(&player2, 950, 20);

SDL\_RenderPresent(renderer);

SDL\_Delay(16);

}

}

**19: Handling Player Input**

Player input is essential for controlling the game. The following function handles the player's movement, jumping, attacking, and defending based on the keyboard keys pressed:

void HandlePlayerMovement(Player\* player, SDL\_Keycode leftKey, SDL\_Keycode

rightKey, SDL\_Keycode jumpKey, SDL\_Keycode attackKey, SDL\_Keycode defendKey)

{

const Uint8\* keystate = SDL\_GetKeyboardState(NULL);

if (keystate[SDL\_GetScancodeFromKey(leftKey)]) { player->vx = -2; player-

>facingRight = 0; }

else if (keystate[SDL\_GetScancodeFromKey(rightKey)]) { player->vx = 2; player-

>facingRight = 1; } else { player->vx = 0; }

if (keystate[SDL\_GetScancodeFromKey(jumpKey)] && !player->isJumping) { player>vy = -10; player->isJumping = 1; }

if (keystate[SDL\_GetScancodeFromKey(attackKey)] && !player->isAttackingLight) { player->isAttackingLight = 1; } if (keystate[SDL\_GetScancodeFromKey(defendKey)]) { player->isDefending = 1; } else { player->isDefending = 0; } player->x += player->vx; player->y += player->vy; }

**20: Methodology:**

The game follows a sequential development approach. The game loop is at the core of the application, constantly updating game state (e.g player position, projectile position) and rendering the updated state on the screen. Player inputs are captured and processed at each frame, ensuring the game responds in real-time.

**21:****Best of Three System in the Game**

The game uses a “best of three” system to decide the winner. This means players need to win two out of three rounds to be declared the overall winner. Each round starts fresh, and the winner is announced after completing the required rounds. This makes the game more exciting and competitive.

**22: Making Sure the Game Prints the Winner**

To ensure the game declares a winner, a condition was added to check the health of both players during the game loop. If a player’s health reached zero, the game would stop and display a message such as “Player 1 Wins” or “Player 2 Wins.” This feature helps players easily understand the outcome of the match.

**23: Maintenance and Update:**

The game is designed for easy maintenance with modular code. Each aspect of the game (player movement, projectile handling, etc.) is separated into functions, allowing developers to easily modify or add new features.

**24: Time and Durations (Project Phases):**

## Phase 1 - Initial Setup

This phase took about one week. The focus was on setting up SDL, creating a basic game window, and preparing the environment for further development.

## Phase 2 - Game Mechanics

This phase lasted two weeks. During this time, we implemented key game mechanics such as player movement, animations, and attack actions, making the game interactive and engaging.

## Phase 3 - Testing and Debugging

The final phase lasted one week. In this phase, we focused on fixing any bugs, optimizing the game’s performance, and refining the animations to ensure everything ran smoothly.

**25: Risk Factors and Challenges:**

* **Player Input Lag**: Handling smooth input recognition is crucial, as any delay in player actions can degrade the gaming experience.
* **Projectile Collision Detection**: Ensuring projectiles are accurately checked for collisions and that the game behaves correctly under various scenarios.

**26: Future Scope:**

* **Multiplayer**: The game could be extended to support online multiplayer features.
* **Enhanced AI**: AI-controlled enemies can be introduced to allow single-player gameplay.
* **Additional Features**: Adding more attacks, player customization, and advanced combat mechanics.

**27: Implementation Strategy:**

The game should initially be tested in local multiplayer mode, followed by optimizations for performance. After the basic game mechanics are solid, new features can be gradually added, ensuring the game remains stable and bug-free.

**28: Unsuccessful Attempt**

During the game development, there was a problem where pressing the attack button sometimes fired two projectiles instead of just one. This happened because the code for

This happened because the code for firing projectiles was running more than once at the same time.

This caused confusion during gameplay, as players could fire multiple projectiles by mistake. The issue was fixed by adding a check to make sure the button could only fire one projectile at a time and would not trigger again until the button was released.

**29: Contribution of Members in the Game**

**Ahmed Hussain**: Worked on the projectile system, including firing, collision detection, and reducing health when a projectile hits a player. He also helped with movement mechanics.

**Shaheer**: Focused on creating and improving character animations, such as walking, jumping, attacking, and defending

**M. Ibrahim**: Debugged the game to fix issues in the game loop, player movement, and animations, ensuring smooth gameplay.

**Shaheer and Ahmed Hussain**: Together, they implemented player controls and movement, allowing characters to walk, jump, and interact with the game world.

**Aliyan Shaikh**: Designed the health bar system to display player health in realtime. Additionally, Aliyan made the game report and other documents, making sure the game was explained well.

**30: Conclusion:**

In conclusion Clash of Arms is a fun and simple 2D fighting game that shows the basics of game development Using SDL and C we created a smooth and interactive experience with features like moving jumping attacking and health management The game reacts to player actions with a health bar that changes when you get hit and smooth animations for walking and attacking

There were challenges like making sure movements were smooth and projectiles hit the right target but by organizing the code well we solved them The game is easy to modify so future improvements like multiplayer or better AI are possible

Overall Clash of Arms is a solid start for a 2D fighting game and has great potential for expansion Its simple now but can grow into something more exciting and with this experience we’re ready to take on more game development projects in the future.

**31: References**

1. SDL Documentation: Simple DirectMedia Layer
2. SDL2: A Beginners Guide - Learn C Programming