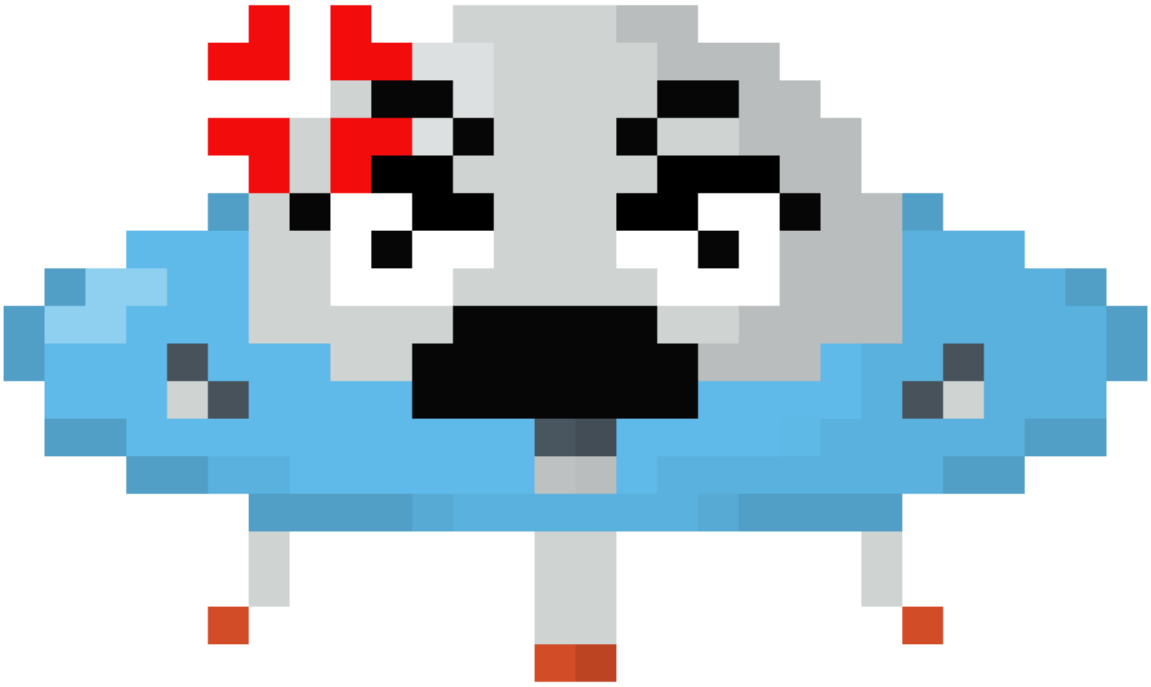
ALIEN EVASION: DODGE EM' ALL

~A C++ Raylib video game~

By Colten L. and Tyson S.



This is a project made using the 2D C++ game library known as Raylib. The core gameplay loop features the player as a spaceship which will rotate according to the mouse cursors position. The player will be attacked by aliens, the player will be able to shoot at them to destroy the aliens. The game will run until the player is killed and a high score will be displayed based on the number of aliens destroyed and time alive.

TABLE OF CONTENTS

1. Introduction/Problem Definition

Problem description and objectives:

The project goal is to make a game that the user can play and interact with. Some objectives include movement for both players and enemies, enemy logic, collision logic, bullet firing and a game over screen + scoreboard for UI.

Development environment:

Hardware:

CPU: Intel(R) Pentium(R) CPU G2030 @ 3.00GHz

RAM: 8GB

Software:

OS: Windows 10

IDE: Microsoft Visual Studio

Installed Programs: Raylib

Operational environment:
Software:
Installed Programs: Raylib, C++

4. Implementation

The games UI will update the score based on the time the player is alive and how many aliens they destroy. The UI will also provide contextual tips to give the rules to the player. The main software needs the handle the player movement via keyboard input, the aliens movement that tracks directly to the player. In addition the player can hold the left mouse button to fire a projectile that will destroy an alien and add to the score.

5. Installation and Operating Instructions

External User Documentation (User guide with installation instructions, training materials, Help files and FAQs, etc.)

- Navigate to the downloadable file
- After installing the file open/unzip
- Locate the {filename}.exe inside of the installed file
- Open the executable to open the application
- CONTROLS
- SPACE key movement to propel forward
- F11 to enter fullscreen
- ESC to close the application

7. Recommendations for Enhancement

- One thing I would've liked to add is the rest of my sprites for the various elements of the game (Tyson)
- An additional feature I would like to add is various type of aliens as well as power ups to change the projectiles or other mechanics of the game (Tyson)
- I would like to fix the issue with speed variables and how they are affected by the screen size (Colten)
- Add better UI, possibly a start screen, and a pause screen. (Colten)

8. Citations

GitHub. (n.d.). https://github.com/features/copilot

Games collection. raylib. (n.d.). https://www.raylib.com/games.html

Cheatsheet. raylib. (n.d.-a). https://www.raylib.com/cheatsheet/cheatsheet.html

Tyson S. Log:

- June 6-7
  - Setup C++ Project file
  - Created final report page
  - Begun drawing sprites to be used for the game

- Brainstormed ideas for implementations of features

- June 10-14
  - Created the alien behaviour (they will spawn at a given rate and chase the player)
  - Assisted in creation of the timer to track how long the player is alive for (See Coltens log for source code)
  - Created sprites for the player, bullets and aliens
  - Assisted in the game over function logic (See Coletens log for source code)

```
// Alien Chaser and Basic Alien Functions, variables and testing ----------
// Alien draw by type function
void drawAlien(Vector2 position, float rotation, Color color, int type)
{

    // Draw the alien
    if (type == 1)
    {
        DrawPoly(position, 4, 20, rotation, color);
    }
    else if (type == 2)
    {
        DrawPoly(position, 4, 20, rotation, color);

    }
}

 // Move alien by type function
void moveAlien(Vector2& position, float& rotation, int type) {
    if (type == 1) {
        //get the direction to the player
        Vector2 direction = { playerPosition.x - position.x, playerPosition.y - position.y };

        //get the angle to the player
        float angle = atan2(direction.y, direction.x);

        //rotate the alien towards the player
        rotation = angle * RAD2DEG;

        //move the alien towards the player
        position.x += cos(angle) * 0.1;
        position.y += sin(angle) * 0.1;
    }
}


// Spawning for aliens of provided type and amount ------------------------------------------------------------
 //Create a struct for the alien chaser and basic alien
struct Alien
{

    Vector2 position;
    Vector2 direction;
    float speed;
    float angle;
    int type;


};


// ------------------------------------------------------------------------------------------------------------

// Alien Chaser and Basic Alien Variables
vector<Alien> chaser;
vector<Alien> basic;
```

```
frameCounter2++; //Frame counter for the alien spawning
if (frameCounter2 % 1440 == 0)
{
    // create a new alien that is not too close to the player
    Alien newAlien;
    newAlien.position = { (float)GetRandomValue(0, screenWidth), (float)GetRandomValue(0, screenHeight) };
    // check if the alien position is too close to the player
    while (CheckCollisionPointCircle(newAlien.position, playerPosition, 100)) //New position if collides with player
    {
        newAlien.position = { (float)GetRandomValue(0, screenWidth), (float)GetRandomValue(0, screenHeight) };
    }
    newAlien.angle = 0; // angle of the alien
    newAlien.type = 1; // type of the alien
    chaser.push_back(newAlien); // add the alien to the vector



}

for (auto& alien : chaser)
{
    drawAlien(alien.position, alien.angle, playerColor, alien.type); // draw the alien
    moveAlien(alien.position, alien.angle, alien.type); // move the alien
}
```

(Alien struct, movement and draw functions, when called in Main (), any number of aliens can be created with any values and behaviours based on the: ”type” variable)

Colten L. log:

- June 6-7
  - Setup the main C++ Project file
  - Began working on the movement functions
- June 9-11
  - Finished the movement functions
  - Added a fullscreen function
  - Created Projectile functions
  - Started working on a particle trail
- June 12 – 14
  - Completed the particle trail function
  - Added a function for a game over screen
  - Added a timer to the game to show how long it's been going
  - Began working on texture loading
- June 17
  - Added all alien code to the main project file
  - Added collision checking for the alien to player and alien to projectile
  - Added a score system
  - Updated the UI
  - Added an easy mode

```
// Function to tell player how to enable easy mode and if it is enabled
void easyModeText()
{
    if (easyMode == false)
    {
        // in top right corner
        DrawText("Press F1 to Disable Easy Mode", GetScreenWidth() - MeasureText("Press F1 to Disable Easy Mode", 20) - 10, 10, 20, timeColour);
    }
    else
    {
        // in the top right corner
        DrawText("Easy Mode Disabled", GetScreenWidth() - MeasureText("Easy Mode Disabled", 20) - 10, 10, 20, timeColour);

    }
}
```

```
// Function to enable easy mode if F1 is pressed
void enableEasyMode()
{
    if (IsKeyPressed(KEY_F1))
    {
        easyMode = true;
    }
}
```

```
// Function to make the game fullscreen
void fullscreen()
{
    // Toggle fullscreen
    if (IsKeyPressed(KEY_F11))
    {
        ToggleFullscreen();
    }
}
```

```
// Function for when lives reach 0
void gameOver()
{
    if (lives <= 0)
    {
        // Draw red overlay
        DrawRectangle(0, 0, GetScreenWidth(), GetScreenHeight(), { 125, 0, 0, 255 });

        // Draw Game Over Text in the center of the screen
        DrawText("Game Over", GetScreenWidth() / 2 - MeasureText("Game Over", 40) / 2, GetScreenHeight() / 2 - 40, 40, WHITE);

        // Button to close the game
        DrawText("Press ESC to close the game", GetScreenWidth() / 2 - MeasureText("Press ESC to close the game", 20) / 2, GetScreenHeight() / 2 + 20, 20, WHITE);

    }
}
```

```
// Function to tell player how to enable easy mode and if it is enabled
void easyModeText()
{
    if (easyMode == false)
    {
        // in top right corner
        DrawText("Press F1 to Disable Easy Mode", GetScreenWidth() - MeasureText("Press F1 to Disable Easy Mode", 20) - 10, 10, 20, timeColour);
    }
    else
    {
        // in the top right corner
        DrawText("Easy Mode Disabled", GetScreenWidth() - MeasureText("Easy Mode Disabled", 20) - 10, 10, 20, timeColour);

    }
}
```

```
// Function to make the cursor invisible and replace it with a crosshair
void crosshair()
{
    // Hide the cursor
    HideCursor();

    // Draw the crosshair
    DrawLine(GetMouseX() - 10, GetMouseY(), GetMouseX() + 10, GetMouseY(), SKYBLUE);
    DrawLine(GetMouseX(), GetMouseY() - 10, GetMouseX(), GetMouseY() + 10, SKYBLUE);
}

// Function to draw collide with cursor
void cursorCollisionCheck()
{
    // Check if the cursor is colliding with the player
    if (CheckCollisionPointCircle(GetMousePosition(), playerPosition, 20))
    {
        // Reset the player position
        //playerPosition = { 400, 400 };
        lives--;
    }
}
```

```cpp
// Function to update and draw particles
void updateParticles()
{
    for (auto it = particles.begin(); it != particles.end();)
    {
        // Update particle position
        it->position.x += it->direction.x;
        it->position.y += it->direction.y;

        // Fade out the particle
        it->opacity -= 255.0f / it->lifeTime;

        // Draw the particle
        DrawCircleV(it->position, particleRadius, Fade(particleColor, it->opacity / 255.0f));

        // Check if the particle is no longer visible
        if (it->opacity <= 0)
        {
            it = particles.erase(it);
        }
        else
        {
            ++it;
        }
    }
}
```

```cpp
// Function to move the player towards the mouse position using spacebar
void movePlayer(Vector2& position, float& rotation)
{
    // Get the mouse position
    Vector2 mousePosition = GetMousePosition();

    // Get the direction to the mouse
    Vector2 direction = { mousePosition.x - position.x, mousePosition.y - position.y };

    // Get the angle to the mouse
    float angle = atan2(direction.y, direction.x);

    // Rotate the player towards the mouse
    rotation = angle * RAD2DEG;

    // Move the player towards the mouse
    if (IsKeyDown(KEY_SPACE))
    {
        frameCounter2++;
        // Accelerate the player
        playerSpeed += playerAcceleration;

        // Limit the speed
        if (playerSpeed > speedLimit)
        {
            playerSpeed = speedLimit;
        }

        // Move the player
        position.x += cos(angle) * playerSpeed;
        position.y += sin(angle) * playerSpeed;

        // Create particles at a slower rate
        if (frameCounter2 % particleTime == 0)
        {
            particles.push_back({ position, { -cos(angle) * particleSpeed, -sin(angle) * particleSpeed }, particleOpacity, 255 });
        }


    }
    else
    {
        // Decelerate the player
        playerSpeed -= playerAcceleration;

        // Limit the speed
        if (playerSpeed < 0)
        {
            playerSpeed = 0;
        }

        // Move the player
        position.x += cos(angle) * playerSpeed;
        position.y += sin(angle) * playerSpeed;

    }
}
```

```cpp
// Function to shoot a projectile
void shootProjectile(Vector2 position, float rotation) {
    Vector2 direction = { cos(rotation * DEG2RAD), sin(rotation * DEG2RAD) };
    projectiles.push_back({ position, direction });
}
```

```cpp
// Function to update and draw projectiles
void updateProjectiles() {
    for (auto it = projectiles.begin(); it != projectiles.end();) {
        // Move the projectile
        it->position.x += it->direction.x * projectileSpeed;
        it->position.y += it->direction.y * projectileSpeed;

        // Check if the projectile is off screen
        if (it->position.x < 0 || it->position.x > GetScreenWidth() || it->position.y < 0 || it->position.y > GetScreenHeight()) {
            it = projectiles.erase(it);
        }
        else {
            // Draw the projectile
            DrawCircleV(it->position, projectileRadius, projectileColor);
            ++it;
        }
    }
}

```

```cpp
// Function to check collision between player and alien given position and radius
bool checkCollision(Vector2 position, float radius)
{
    if (CheckCollisionPointCircle(playerPosition, position, radius))
    {
        lives--;
    }
    return false;
}
```

```cpp
// Function to draw the player as a poly
void drawPlayer(Vector2 position, float rotation, Color color)
{
    // Draw the player
    DrawPoly(position, 3, 20, rotation, color);
}
```

Software Development Report Outline Example

Title Page (Title of the project, Group member names)

Abstract (Keep it short!)

Table of Contents (Including tables, figures, and appendices if any)

1. **Introduction/Problem Definition**

    a. Problem description and objectives: State your project goals and what was accomplished and why it was significant enough to justify a development project. Describe the "market", or "user demand", for your software.
    b. Development environment: software and hardware
    c. Operational environment: software and hardware

2. **Requirements and Design Description** (architecture, external and internal functions, interfaces)

    Supplementary written description/analysis of all models (UMLs, flowcharts, user interface).
    Brief descriptions to accompany each model.
    New Learning: UML class diagrams: read and/or listen.

4. **Implementation**

    The program objectives and end user requirements. In other words, what your software is capable of and what the user needs to be able to achieve when using the product. A brief explanation/pseudocode to outline it. You may need separate sections for each member of your groups' work.

5. **Installation and Operating Instructions**

    External User Documentation (User guide with installation instructions, training materials, Help files and FAQs, etc.)

7. **Recommendations for Enhancement** (What could be improved?)

8. **Bibliography** (Cite all references used, APA style)

Appendix A: UML Diagrams, system flowcharts

Appendix B: Images of the user interface

Appendix C: Internal Documentation (your code with comments, docstrings, block/line comments etc.)

https://www.programiz.com/python-programming/docstrings

Appendix D: Individual student work logs.

Eg: A log book or diary of activities, work completed, etc.; Any evidence that the student has contributed positively to the project; e-mailed documents, progress reports, discussions, aha moments etc.

Sample Table of Contents

Final Software Product (15% of summative)

Submit source files via dropbox or USB (fully documented program code to include doc comments, docstrings, and some block comments and line comments). This will be evaluated on the basis of the quality of the product developed and the thoroughness of the individual tasks performed by each member.

Formal Report (10% of summative)

See template above for requirements.

Group Presentation of Product (5% of summative)

Demo functional program for the class.
Lead the class through the development process and highlight challenges and successes along the way.