<u>Program Analysis:</u>

This Program is an arcade style airplane simulation. The user will control an airplane with the mouse, to fire on other enemy airplanes on the screen. This program will be done using graphics, to draw the airplanes, as well as the ground below and the gunfire. Data storage needed will include a class to contain the user's plane information, such as position, health, as well as functions needed to draw itself on the screen, and detect whether it has been hit by an enemy. Other classes needed will be that of the enemy aircraft, which will be similar to the player class, containing data for its position, but also include the direction it will move itself across the screen, without player input. Next classes will manage the background terrain that wills scroll under the aircraft, which need to posses data for what type of terrain it should display and what its potion on the screen should be. Another class will also be needed, in the form of a linked list to track the various bullets that will have to move across the screen during operation.

<u>Design Process:</u>

<u>Initial Design:</u>

```
1. Declare Window
2. Declare Player
3. Declare Map
    a. Read Map from text file
    b. Insert into matrix
4. Declare Enemies
5. Declare list of Bullets
6. While the right mouse button is not clicked, or the player is not dead…
    a. Terrain cycle
        i. Move Terrain
        ii. Draw Terrain
    b. Player Cycle
        i. Detect if player is hit
            1. if hit, explode, and delete bullet from list
        ii. Move Player
        iii. Detect if mouse button is down
            1. If it is, Fire, and add a new bullet to the list of bullets
        iv. Draw Player
    c. Enemy Cycle
        i. Detect if enemy is hit
            1. If hit, explode, and delete bullet form list
        ii. Move Enemy
        iii. Determine whether the enemy will fire
            1. If it fires, add a new bullet to the list of bullets
        iv. Draw Enemy
    d. Bullet Cycle
        i. Move each bullet
        ii. Delete bullets that are off the screen
        iii. Draw Bullets
    e. Update graphics buffer
7. End program
```

Data Structures:

Class Player

Player(window & tempWindow, BulletList & tempBulletList, EnemyList & tempEnemyList)

Variables

int xPos
int yPos
int Width
int Height
int ReloadTime
int DeathState
int Health
bool isDead
unsigned long LastShot
window *myWindow
BulletList *myBulletList
EnemyList *myEnemyList

Functions

void Draw()
void DrawHealthBar(int score)
void Move()
void Fire()
void CheckHit()
virtual ~Player()

Class Ground

window *myWindow
EnemyList *myEnemyList
int yPos
int yLoc
bool atEnd; //Flag when level is completed

char Terrain[15][200]
char Enemies[15][200]
image water
image grass
image trees
image roadup
image roadside
image beach

Functions

void Declare()

void Draw()
void Move()
void DrawSquare(int x, int y)
void ReadFile()
void ReadEnemies()

<u>Class EnemyList</u>

EnemyList(window & tempWindow, BulletList &tempBulletList)
virtual ~EnemyList()

<u>Variables</u>

window *myWindow
BulletList *myBulletList
EnemyNode *myHead
int NumKilled

<u>Functions</u>

void Move()
void MoveOne(EnemyNode * tempNode)
void Draw()
void DrawOne(EnemyNode * tempNode)
void RemoveEnemy(EnemyNode * tempNode)
void AddEnemy(int x, int y, int DownSpeed, int RightSpeed)
void CleanUp()
void CheckHit()
int NumEnemys()
void Fire()

<u>Class EnemyNode</u>

Enemy enemy
EnemyNode *next
EnemyNode *prev

<u>Class Enemy</u>

Enemy()
virtual ~Enemy()

<u>Functions</u>

void Declare(window & tempWindow, BulletList & tempBulletList, int x, int y, int Right, int Down)

void Move()
void Draw()
void Fire()
void CheckHit()
bool OutofBounds()

<u>Variables</u>
window *myWindow
BulletList *myBulletList
int Height
int Width
int xPos
int yPos
int RightSpeed
int DownSpeed
int ReloadTime
long unsigned LastShot
int DeathState

<u>Class BulletList</u>

BulletList(window & tempWindow)
virtual ~BulletList()

<u>Variables</u>
window *myWindow
BulletNode *myHead

<u>Functions</u>
void Move()
void MoveOne(BulletNode * tempNode)
void Draw()
void DrawOne(BulletNode * tempNode)
void RemoveBullet(BulletNode * tempNode)
void AddBullet(int x, int y, bool isEnemy, bool isUp)
void CleanUp()
int NumBullets()

<u>Class BulletNode</u>

Bullet bullet
BulletNode *next
BulletNode *prev

<u>Class Bullet</u>

Bullet()
virtual ~Bullet()

<u>Variables</u>
bool isUp
bool isEnemy

```
int xPos
int yPos
int Speed
int Height
window *myWindow
```

Functions

```
void Draw()
void Move()
void Declare(window & tempWindow, int x, int y, bool Enemy, bool Up)
```

Algorithms:

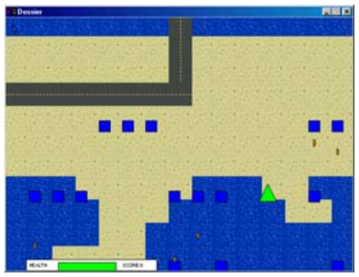
All the algorithms in this program are efficient, do not use more memory than is need, nor do they take excessive amount of computing power. Each function is designed in a way to interact with other classes in a timely manner, in order to keep the real time graphics updated for the user. Without efficient algorithms, this would not be possible. The algorithms are also created in a way that allows changes to be made easily, using loops, recursion, and dynamic data when possible.

Testing Strategy:

Test Run	Data	Reason	Expected Results
1	Input unexpected keyboard commands: Escape, Spacebar, F1, CTRL, and Enter.	To ensure the program ignores unexpected input	The program to continue without interruption
2	Remove level.txt from directory	To test that program will operate without it	Program will run with no enemies, and a terrain of all water.
3	File level.txt with less data than is expected, less than 200 rows	To make sure it will not crash, or display unexpected graphics	It will display water squares were data is missing, and not display enemies when data is missing
4	Fill level.txt with data more data than is expected (more than 200 rows of data)	To make sure it will ignore the excess data.	Excess data will be ignored.
5	Fill level.txt with bad data, with characters that are not expected like ‘z’ ‘q’ ‘#’ and other random characters.	To make sure it will operate regardless of bad data	Program will replace all bad data for terrain with water squares, and not create enemies when bad data is read.

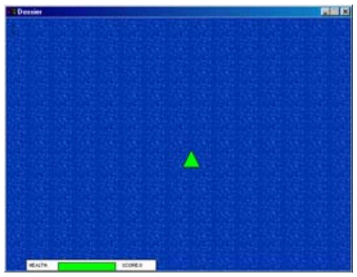
Test 1:

Result: No interruption occurred when pressing buttons.



Test 2:

All terrain is displayed as a water square as expected. No enemies appeared as well.



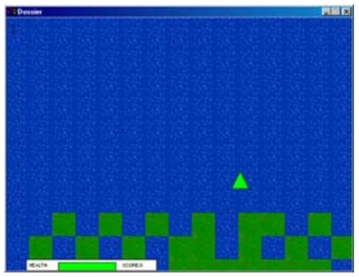
Test 3:

All the terrain that was define in the text file was displayed, and a water square represented the rest. No enemies appeared because the it was interpreted as the missing terrain data. The following is what level.txt contained

<u>Level.txt:</u>

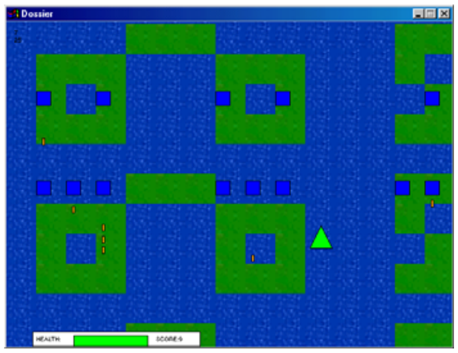
```
111111111111111
000100001001001
000000000000000

101010111111101
101010110101010
010101010110100
```



Test 4:

The Program operated normally; the extra data in the file was ignored, while the rest was used.



Test 5:

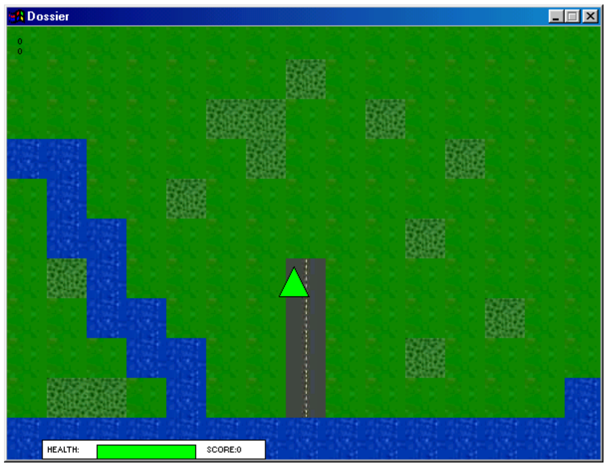
First five rows of level.txt:

```
azxcasdfeoilpqw
qwertyasdsafg;z
!@#$%^%&*(){}\;
<>?:"{}|,./;'[]
13310114111111\
```

Rows 201-206, the first five rows for Enemy data

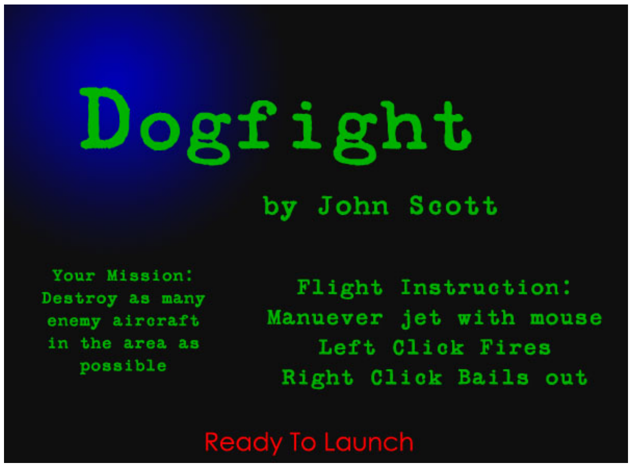
```
!@#$%^&*()_+ads
asdfsadlkfjruoi
<>?:"{}|\]['/.
">gdsfg';;;dfae
000011100000011
```

The result is that the first rows for the terrain are simply water squares, and the first rows for the enemies do not cause enemies to appear



User-Friendly Features:

The program was created to be as easy to use as possible. All data issues are left up to program, without any necessary input from the user. The controls of the program are clearly stated at the introduction screen, in a clear manner. It gives the objective to the user, as well as how to operate the program. All data displayed for the user during the program is clearly labeled for easy readability.



Dogfight

by John Scott

Your Mission:
Destroy as many
enemy aircraft
in the area as
possible

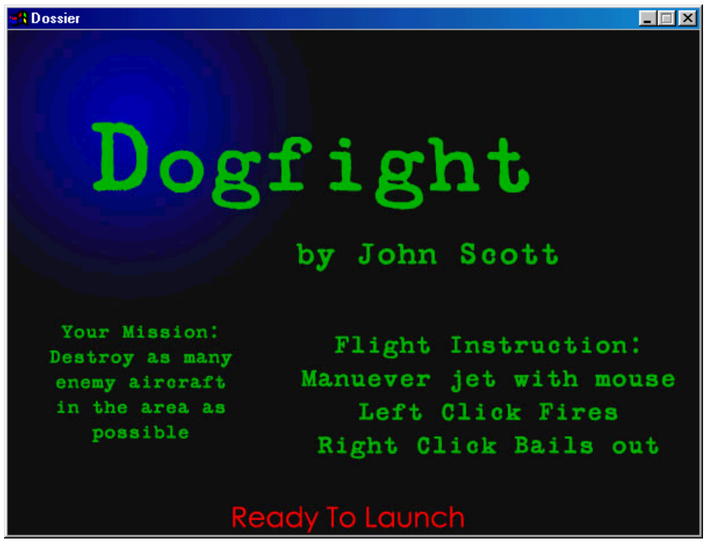
Flight Instruction:
Manuever jet with mouse
Left Click Fires
Right Click Bails out

Ready To Launch

Error Handling

The method of operation used in this program lends itself to excellent error handling. The mouse is used to operate this program, so all keystrokes, or other input devices are completely ignored. Any unexpected mouse clicks are simply ignored, leaving no opportunity for errors to occur from bad input. The data file, which stores the background terrain location information, is also protected from error. Should this file be missing or corrupt, all data missing will simply be replaced by either a water square for the terrain, or if it is with the enemy location data, it will ignore the bad data, without interruption of the program.

<u>Sample Run:</u>
When the program is initially run, the Title screen is displayed


Dossier

Dogfight

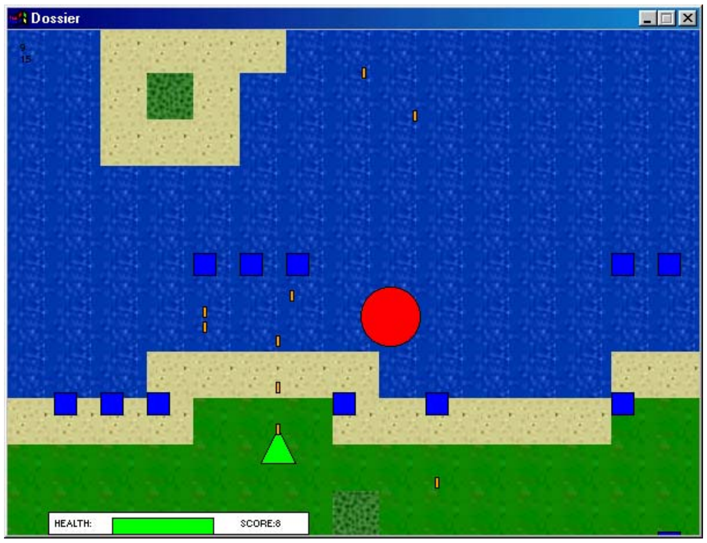
by John Scott

Your Mission:
Destroy as many
enemy aircraft
in the area as
possible

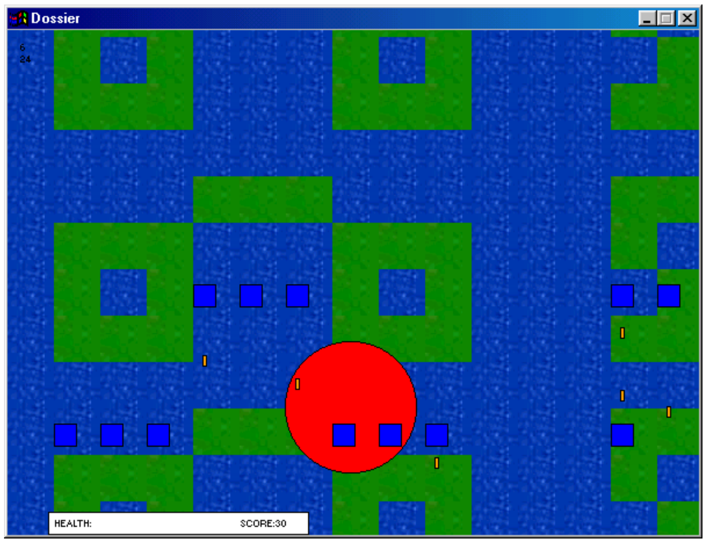
Flight Instruction:
Manuever jet with mouse
Left Click Fires
Right Click Bails out

Ready To Launch

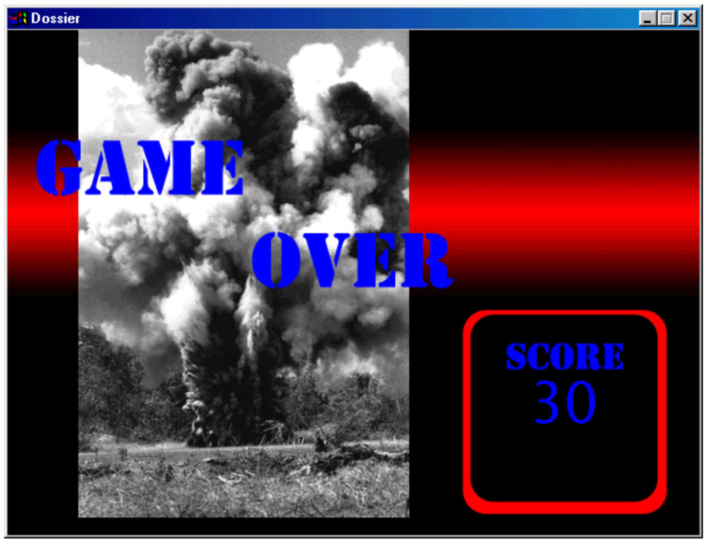
After left-clicking, the program begins to display the terrain, the player's airplane, and the enemies. When left-click is held down, bullets appear from the players aircraft. When they collide with the blue squares, the bullets disappear, and the enemy explodes. Gradually as the green triangle collides with the squares, or is hit by a bullet, it loses health.



Eventually, the player's health reaches zero and the craft begins to explode.



When the game over screen is displayed, the score is printed. One more left click will terminate the program.



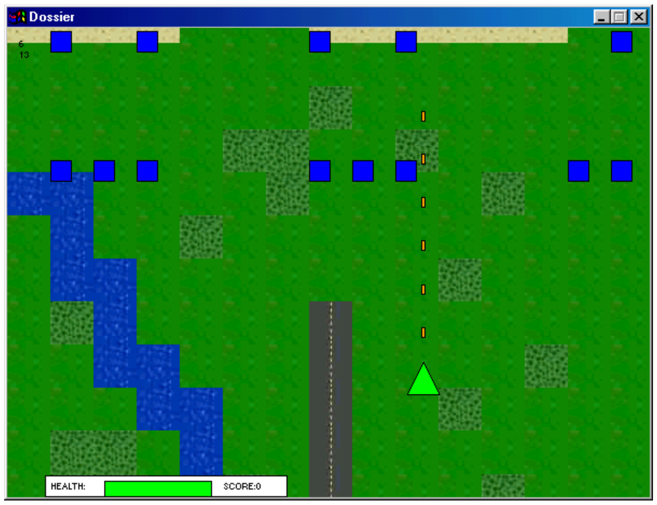
<u>Evaluation:</u>

The final program operates in a very efficient manner. The error handling is superb, the algorithms are efficient, and the program operates with little flaw. The solution made good use of object oriented programming to prevent the program from becoming unreadable. Classes were used for each object both to maintain good programming style, as well as to allow easy modification of the program, to fix any errors or inefficiencies found later on. In some cases, too many classes were used, for example, for the three classes necessary to represent the series of enemies, or the bullets. A better solution perhaps would have been to simply combine the class BulletNode with class Bullet, and do the same with EnemyNode and Enemy. Regardless, either implementation would have had the same effect on the final compiled program.

<u>User Documentation</u>:

Before attempting to run the program, make sure that the following files are all in the same directory:

Dogfight.exe
Level.txt
Beach.jpg
Trees.jpg
Water.jpg
Grass.jpg
Title.jpg
Gameover.jpg
Roadup.jpg
Roadside.jpg



Next, double-click on the icon labeled Dogfight.exe. A title screen should appear, with instructions on how to operate the airplane. When ready to proceed click the any mouse button. Now the screen should display a green triangle at your cursor with scrolling terrain scrolling down the screen. The green triangle is your airplane. You can control it by moving the mouse. Soon blue squares should appear from the top of screen. These are the enemy aircraft. Left click to shoot a bullet at an enemy. Avoid also colliding the green triangle with another enemy, or hitting one of their bullets, as each of these will deplete the health bar labeled at the bottom left corner of the screen. If this bar depletes completely, your plane will explode, and you will be presented with the game over screen. Should at any time during the program you wish to exit, simply click the right mouse button and it will bring you to the game over screen. Then simply click once more and the program will terminate.