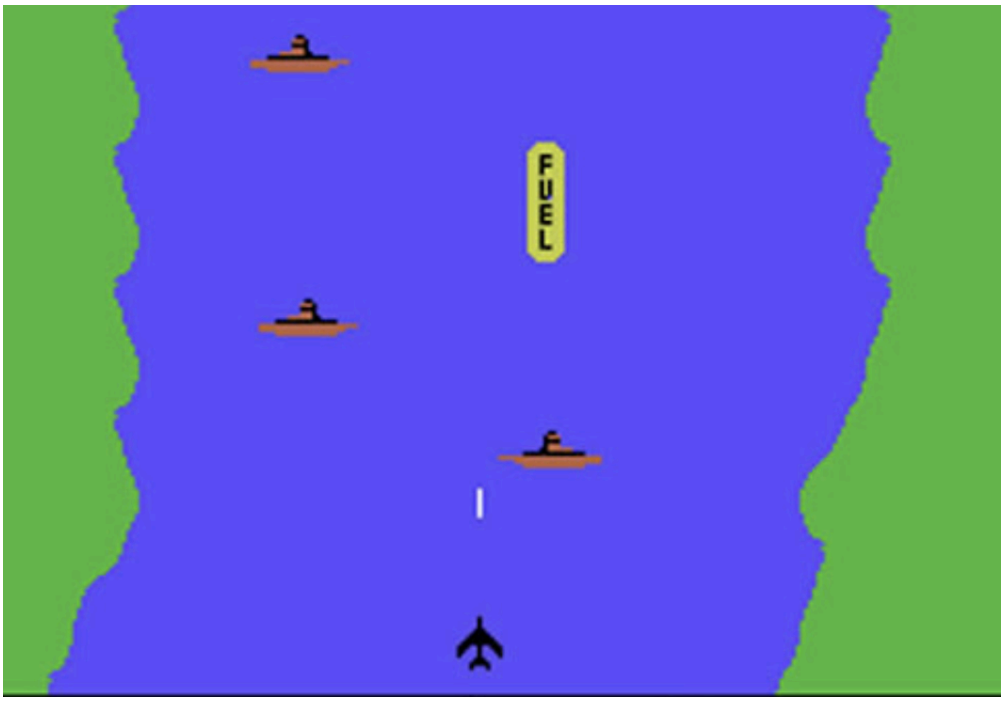
Course Project Statement

Spring 2025

River Raid



Dr. Tamer Ashour Ali

Eng. Ahmed Sherif

Eng. Mohamed Sherif

Eng. Riham Hussain

Eng. Noran Mansour

Introduction

In this project, your task is to develop an engaging **River Raid Game** using object-oriented programming. The game will start by loading or creating a specific map. Then players can start the game in a new window, aiming to score as many points as possible while avoiding obstacles and losing lives.

Gameplay Overview

- You control a jet flying over a winding river.
- The goal is to shoot enemy ships, helicopters, and bridges while avoiding crashing into the riverbanks.
- You need to refuel by flying over fuel depots.
- The game gets harder as you progress, with narrower paths and more enemies.

For a deeper understanding of the game mechanics and features, please refer to the rest of this document.

Project Schedule

Phase	Individual v.s. Team	Deliverables	Due Week	Weight
Team Formation	Team	Fill the form	Week 4	-
Progress Report 1	Individual	Progress Form	Week 6	5%
Progress Report 2	Individual	Progress Form	Week 10	5%
Phase 1	Individual : Team 2 : 1	See the "Project Phases" section	Week 11	30%
Peer Review 1	Individual	Peer Rev Form	Week 11	2.5%
Progress Report 3	Individual	Progress Form	Week 13	10%
Phase 2	Individual : Team 2 : 1	See the "Project Phases" section	Week 15	30%
Final Demo	Team	Final Demo with Phase 2 Discussion	Week 15	15%
Peer Review 2	Individual	Peer Rev Form	Week 15	2.5%

Requirements

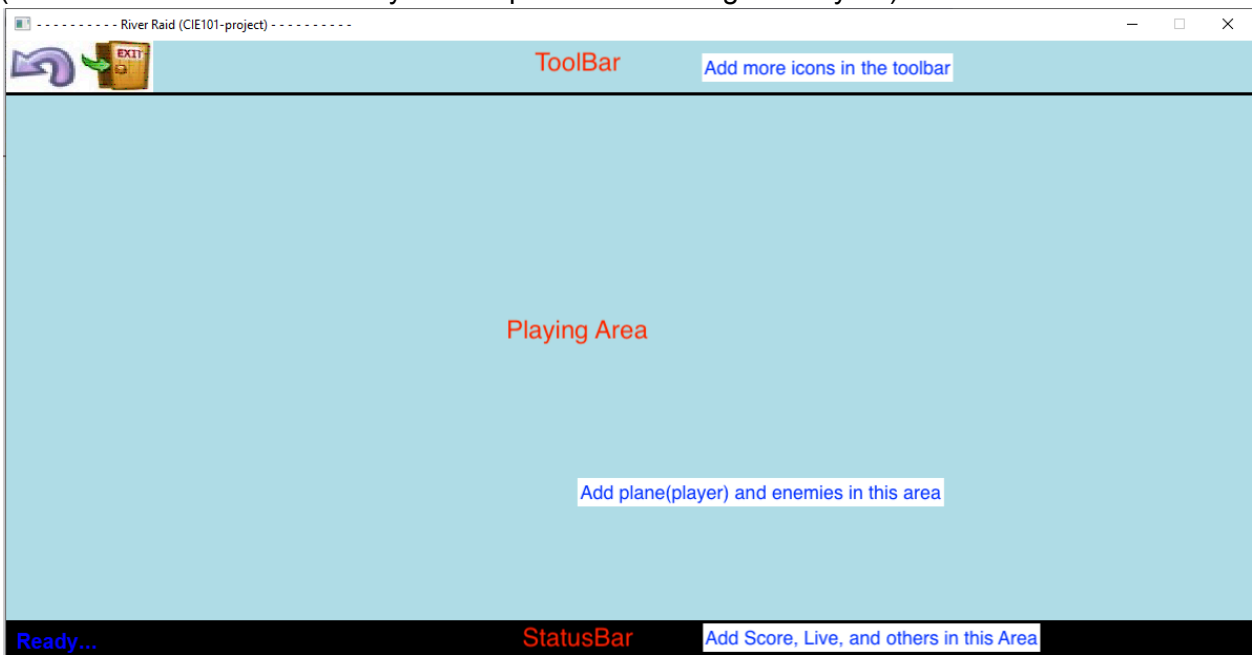
#	Feature	Relative Weight	Phase
1	Draw status bar texts at the bottom of the screen for: -- Points: Starts at 0 -- Game Speed: Starts at 5 -- Lives: Starts at 5 -- Fuel Gauge: Starts at 50	2	Phase 1
2	Draw toolbar icons at the top of the screen for: -- Restart -- Pause -- Resume -- Save -- Load	1	Phase 1
3	Implement a function to draw the player's plane at the center-bottom of the screen	2	Phase 1
4	Implement a function to draw each enemy type: -- Enemy Ships (Tankers) -- Bridges - span across the river.	3	Phase 1
5	Implement a function to draw each enemy type: -- Helicopters -- Jets (Enemy Fighters)	3	Phase 1
6	Implement a function to draw the Fuel Depots.	1	Phase 1
7	Implement a function to draw the bullet.	1	Phase 1
8	Implement a function to draw the background, 3 rectangles with saved positions.	2	Phase 1
9	Implement a function that generates random enemies based on the current level/speed.	2	Phase 1
10	Ensure all draw functions are called in the correct places within the game loop.	1	Phase 1
11	Implement a Move function to loop and animate the background moving downward.	2	Phase 1
12	Implement the collision detection function between any two collided objects.	4	Phase 2
13	Move the plane left and right using the arrows or AD keys.	2	Phase 1
14	Move the plane up (Increasing the speed of the plane), and down (Decreasing the speed) using the arrows or WS keys.	2	Phase 1

#	Feature	Relative Weight	Phase
15	Implement a function to launch bullets vertically when the space key is pressed.	2	Phase 1
16	Delete bullets when they reach the top boundary (before the toolbar).	1	Phase 2
17	Implement Enemy Movement Functions: -- Tankers: Move vertically along the river. -- Helicopters: Move side-to-side across the river. -- Jets: Move horizontally across the river. -- Bridges: Stationary targets. -- Fuel Depots: Doesn't move.	4	Phase 2
18	Implement a function to delete enemies once they move off-screen or are destroyed.	1	Phase 2
19	Ensure all move functions are called in the correct places within the game loop.	2	Phase 2
20	Destroying Enemy Units - Collision Action -- Tankers (Ships): +30 points -- Helicopters: +60 points -- Jets (Enemy Fighters): +100 points -- Bridges: +500 points (highest single-action reward)	2	Phase 2
21	Implement Pause / Resume icon actions.	2	Phase 2
22	Implement Restart icon action.	1	Phase 2
23	Implement the save icon action: -- Save the progress of the current game.	3	Phase 2
24	Implement the load icon action.	3	Phase 2
25	- Initially, enemies appear sparsely and move slowly. - As the player advances, enemies spawn more frequently and move faster, making it harder to react.	2	Phase 2
26	- At the start, fuel depots appear frequently, making it easy to replenish fuel. - Later in the game, fuel depots become less common, forcing the player to be more strategic about fuel management.	2	Phase 2
27	- The game's scrolling speed gradually increases, forcing quicker reactions.	2	Phase 2

#	Feature	Relative Weight	Phase
28	Lives decrease when the player crashes into enemy units, bridges, or riverbanks, or runs out of fuel. - When all lives are lost, the game is over.	3	Phase 2
29	Fuel decreases when the jet is flying; fuel constantly drains over time.	2	Phase 2
30	Shooting a fuel depot destroys it, preventing the player from refueling later.	2	Phase 2
31	Add Sound Effects and Background Music	1	Bonus
32	Introduce Boss Enemies at the end of every 3rd level with higher health and unique attack patterns	2	Bonus
33	Prompt the player to enter a username at game start. Track and store their highest score. At game over, update and display a leaderboard showing top scores for all users.	2	Bonus
34	Gradually change the background appearance over time. Instead of using only three static rectangles, vary the river's width and add obstacles or structures in the middle to increase difficulty and visual variety.	2	Bonus
35	Multiplayer Mode (Split Screen or Turn-Based) allowing two players to compete or co-op	4	Bonus
36	You may add additional bonus features with TA approval, but they must introduce something entirely new.		Bonus

Game Screenshot

(This screenshot was drawn by a startup code that was given to you)



The game window is divided into three areas:

- Toolbar: containing icons for save, load, pause,.... etc
- Playing Area: where the user can add the plane, and enemies and move them.
- Status bar: to display messages about score, lives, …etc to the user.

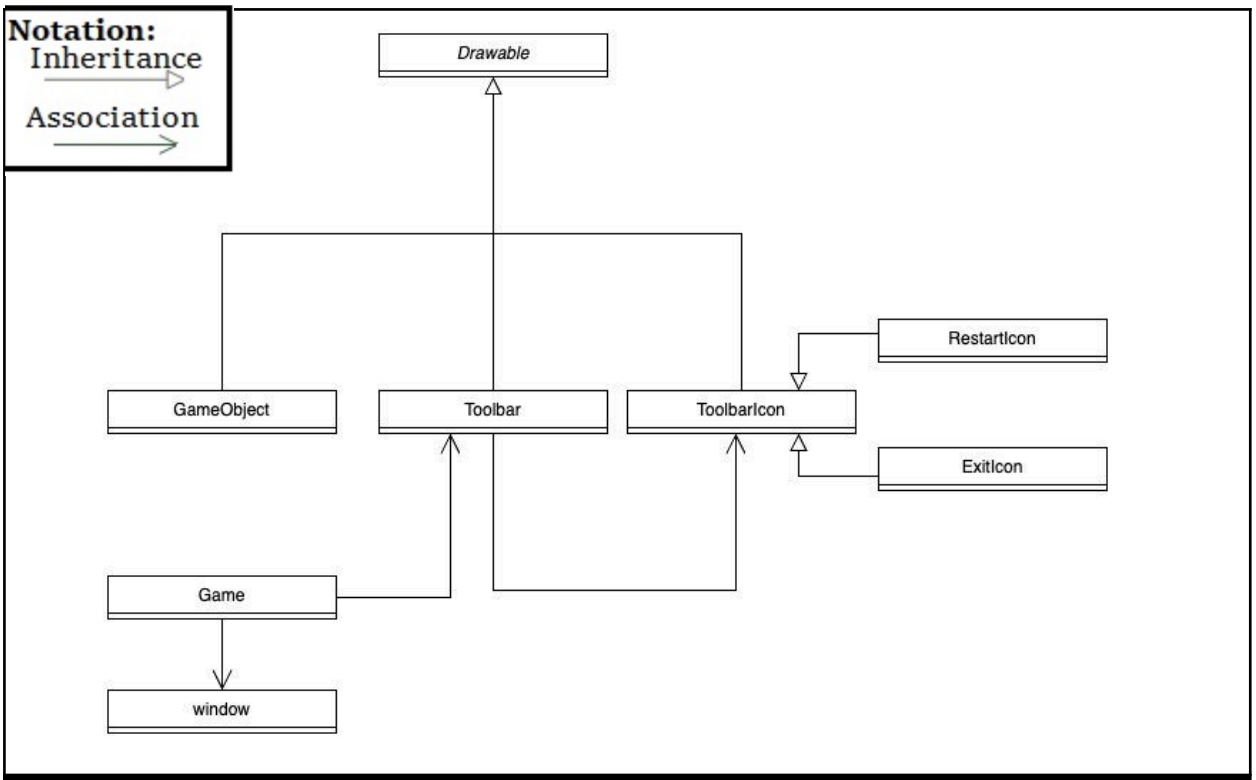
Main Classes

Since this is your first object-oriented programming project, we've provided you with a code framework that includes some partially implemented classes.

For the graphical user interface (GUI), we've integrated the open-source CMU Graphics Library, which will help you create and manage the GUI more easily.

- Your task is to follow the given design and complete the framework by:
- Extending existing classes,
- Inheriting from them,
- Or, if needed, creating new base classes — but only after getting approval from your instructors.

Below, you'll find the class diagram followed by a description of the main classes.



Class Diagram

Drawable Class:

This is the base class for any object that appears on the screen, like the toolbar, the plane, or the enemy helicopter. It contains:

- The **x and y** coordinates of the object's position.
- A **pointer** to the **Game** object, which gives access to the window where everything is drawn.
- An abstract function draw() that must be overridden by any class that inherits from Drawable.

GameObject Class:

This class represents drawable objects that can:

- Have border and fill colors.
- Move around.
- Detect and respond to collisions.

When two GameObjects collide, an action should happen. Each object handles this by overriding the virtual function collisionAction() from the base class.

Toolbar Class:
This class manages the toolbar at the top of the game, which contains icons like Restart, Pause, etc.

ToolbarIcon Class:
This class manages all toolbar icons. It detects which icon the user clicks and calls that icon's onClick() function.

- Each icon should be a subclass of ToolbarIcon. For example, we've already implemented a RestartIcon that restarts the game.
- You should add more icons, and for each one, override the onClick() function to define what happens when it is clicked.

window Class:
This class handles everything the user sees or interacts with. All input and output (drawing on the screen, getting user clicks, etc.) goes through this class.
If any other class wants to show or read something from the user, it must call a function from Window.

Game Class:
This is the main class that manages the whole app.

- It holds pointers to the Toolbar and Window and controls the flow of the game.
- It does not do everything itself — instead, it tells other classes what to do and when to do it.

File Format

Save and Load Feature Requirements (Features 23 & 24):
Your save/load functions must store and restore the full game state when the player clicks Save or Load. Use a plain **.txt** file to save all necessary data, including:

- Game progress: level, points, fuel, lives, speed
- Player plane position
- All active enemies: position, type (e.g., 1 = jet, 2 = tanker), and any extra data required to recreate them (e.g., width, height, or other custom attributes used in your logic)

You may include additional data as needed based on your implementation. You can modify the order of saved items, but you must load them in the exact same order they were saved.

- Here is a sample file, but you're free to adapt it to fit your design.

```
3     // Level
750   // Points
42    // Fuel
2     // Lives

120   // PlaneX
430   // PlaneY

3     // Number of Enemies

1 200 100 40 40   // Enemy 1: Type X Y Width Height
2 180 250 60 60   // Enemy 2: Type X Y Width Height
3 150 300 100 20  // Enemy 3: Type X Y Width Height

150   // Background Scroll Offset

2     // Number of Bullets
120 300   // Bullet 1: X Y
122 310   // Bullet 2: X Y
```

Project Phases Guidelines

Progress Report:

Your progress report should comprehensively reflect your team's work over the past two to three weeks.

1. Begin by providing fundamental information such as your team's name and members.
2. Following this, list the distribution of features amongst students during the reporting weeks.

For each team member, the report should include:

- Planned Goals: Outline the objectives that were set at the beginning of the period.
- Accomplished Goals: Detail the tasks and features that have been successfully completed.
- Blocked Goals (if applicable): Discuss any objectives that could not be achieved and explain the reasons for the same.
- Challenges: Share the difficulties encountered during the work process.
- Lessons Learned: Convey new knowledge or insights gained during this period.
- Future Goals: Set clear objectives to be accomplished in the upcoming weeks, providing a roadmap for your project's next steps.

Phase 1:

Refer to the section titled "Requirements" above, where necessary features for this phase are explicitly highlighted. Tasks and responsibilities should be equitably distributed among team members to ensure a **fair** workload.

Submission Guidelines:

Teams must submit a single zipped file named ProjectPhase1_T# with the following contents:

1. Info.txt: A file containing essential team information such as Names, IDs, and Emails.
2. Workload Division Document: Include a table listing each team member alongside the respective features they have implemented.
3. Code/Resources Files: Ensure that files related to the required features are adequately included.

Phase 2:

Refer to the "Requirements" section above, where necessary features for this phase are explicitly highlighted. Tasks and responsibilities should be equitably distributed among team members to ensure a fair workload.

Submission Guidelines:

Teams must submit a single zipped file named ProjectPhase2_T# with the following contents:

1. Info.txt File: A file that has your team's basic information like names, IDs, and emails.
2. Project Files: Put all the important project codes and files here.
3. Sample Graph (Game Snippet) Files: Please give us three different graph examples. For each graph:
    a. Give a text file that talks about the graph.
    b. Show a picture of the graph made by your program.
    c. Show a picture where the graph is being used or played with.

Grading Criteria for Phases 1 and 2:

The grading for each feature during these phases will be split into two main categories:

- Individual Grade: Each student will receive a grade based on their contribution to the implementation of specific features.
- Team Grade: An overall team grade will also be assigned, reflecting the integration of individual features into the cohesive whole of the project.

Late Submission Policy

- For Phase 1:
  - You can submit late, but only up to 3 days after the due date.
  - If you submit it late, your score will be lower. Each late day will take away 10% of your score.
  - After 3 days, no more late submissions will be accepted.

- For Phase 2 and the Final Parts: No late submissions are allowed. Make sure to submit it on time.

Penalties

Finding	Penalty in Phase 1	Penalty in Phase 2	Affected Part
Undoubted plagiarism	x 0.0	x 0.0	Entire project
Non-working / Unreachable code	x 0.8	x 0.5	Per Feature
Crashing code	x 0.9	x 0.8	Per Feature
Violation of OOP Concepts	x 0.9	x 0.5	Per Phase
Memory Leaks	-	x 0.9	Entire project
Deferring a Phase 1 feature to Phase 2 *	x 0.5	-	Per Feature

NOTE

For Phase 1, Buggy/ Fully Missing Features:

- You can deliver features that were buggy or entirely missing in Phase 1, along with Phase 2.
- However, the maximum grade you can earn for such a feature is **50% of the original** allocation for that feature in Phase 1.