National University of Computer and Emerging Sciences
FAST School of Computing Spring 2024 Islamabad Campus

Operating Systems (CS)

Project Title: Multi-threaded Pac-Man

Deadline: Sunday, 12th May, 2024 (11:59 PM PST)

Total attainable Marks: 160 (Plus Bonus: 30)

- **The project can be done in a group of one/two students only (maximum 2 students only). Please don't ask for groups of three. You are allowed to make cross-section groups too.**
- Zero marks will be awarded to the students involved in plagiarism.
- All the submissions will be done on Google classroom.
- You have to submit .c files in the Zip Folder named after your roll no (20I-XXXX.zip). Naming convention has to be followed strictly.
- Be prepared for demo after the submission of the project.

Project Objective:

Develop a multi-threaded version of the classic Pac-Man game using synchronization techniques to manage interactions between Pac-Man and multiple computer simulated ghosts. The game should handle data conflicts and ensure smooth gameplay with each entity operating on its own thread.

Programming Language: C
Graphics Library: GLUT or SFML

Note: For including graphics in your code, we have provided the skeleton code of GLUT library only. As the students have reported that they have experience of graphics using SFML. So all of you have a free choice to use either of GLUT or SFML. Just remember that SFML is probably a bit higher level than GLUT. You can experiment with both libraries to determine which is more suitable for multi threading project considering the pthread library and other system calls you will need in your project.

Project Description:

Phase 1 - Setting up Dedicated Threads for Game Modules:

Game Engine Thread:

- Create a dedicated thread for the game engine, which is responsible for coordinating the overall game flow, handling input from players, updating the game state, and rendering graphics.
- This thread will execute the main game loop, continuously updating the game state based on user input and Ghost movement controller.

User Interface Thread:

- Implement a separate thread to manage the user interface (UI) components of the game, including menus, scoreboards, and any Heads-up display (HUD) elements.
- This thread will handle input events from the player, such as keyboard or mouse input, and communicate them to the game engine thread for processing.

Ghost controller Threads:

- Create individual threads for each ghost controller responsible for controlling the behavior of ghosts in the game. There will be four ghost threads in the beginning of the game. There may be an increase in the number of ghosts as the game time progresses.
- Each ghost controller thread will execute its ghost algorithm independently, determining the movement patterns and actions of the ghosts based on the current game state. If you are developing a slightly intelligent ghost, it should calculate a simple shortest path algorithm to go in the direction of the Pac-man. Alternatively, for a simpler version each ghost can follow a pre-defined path set for itself at the time of creation of thread.

Phase 2 - Basic Game Mechanics:

Game Board Initialization:

- Design and implement the layout for the **shared** game board among all threads, including walls, paths, pellet positions, and power pellets. All these items in the game board should be visible / accessible to the concerned threads.
- Initialize the starting positions for Pac-Man and the ghosts. The PacMan can start always at a predefined location on a board. The ghost house in the center of the game board is where ghosts start and return to after being eaten. The entrance to the ghost house is a shared resource that only one ghost can use at a time.

Movement Mechanics

- Implement the movement logic for Pac-Man, allowing user input to dictate direction.
- Design basic AI for ghost movement as defined above in the Ghost controller thread section, allowing them to roam the board on their paths.

Eating Mechanics

- Enable Pac-Man to eat pellets as he moves over them, updating the game score.

- Define the effects of Pac-Man eating a power pellet, such as ghosts turning blue for a limited time.

Lives

- Implement a lives system for Pac-Man, allowing the player to have multiple attempts at completing the game.
- Define the initial number of lives for the player and decrement this count each time Pac-Man collides with a ghost.

Phase 3 - Synchronization:

Scenario 1 - Ghost Movement: This scenario remains the same. Ghosts need to read the game board to decide their next move. Pac-Man needs to update the game board by eating pellets. Students will need to ensure that no read operation happens during a write operation.

Scenario 2 - Eating Power Pellets: This scenario also remains the same. When Pac-Man eats a power pellet, the ghosts turn blue and can be eaten by Pac-Man. The power pellets are eaten by Pac-Man and respawned by the game. Students will need to ensure that no two power pellets are eaten at the same time and that no power pellet is eaten when none are available.

Scenario 3 - Ghost House: Each ghost needs two resources to leave the ghost house: a key and an exit permit. The keys and exit permits are limited. Students will need to ensure that no deadlock situation occurs when all ghosts try to leave the ghost house at the same time.

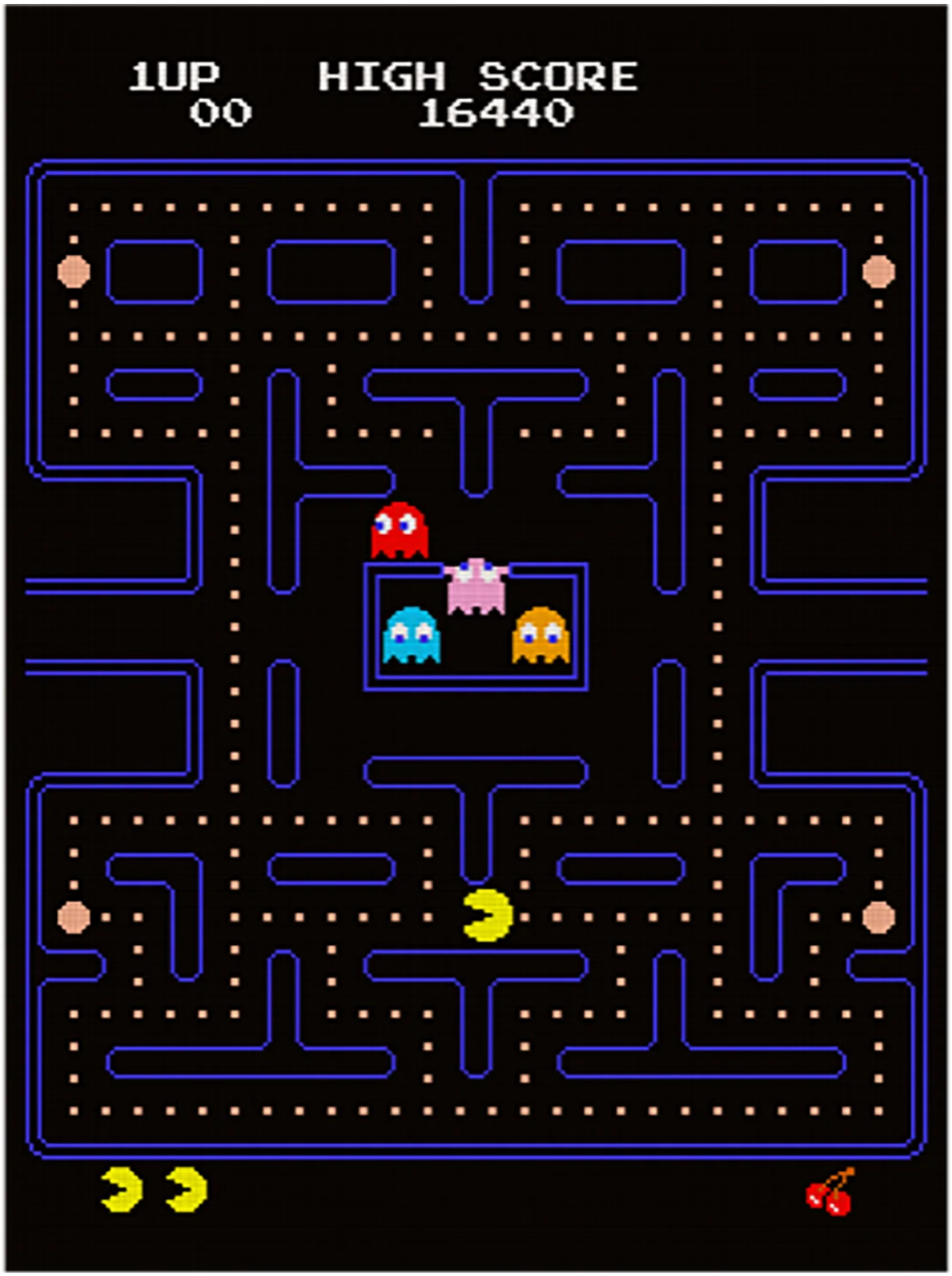
Scenario 4 - Prioritizing Certain Ghosts: Certain ghosts are faster than others. The faster ghosts need an extra speed boost ingredient from the game. The game has a limited number of speed boosts i.e. only 2 speed boosts that can be used by any two ghosts. Each speed boost can only boost one ghost at a time. Students will need to ensure that all faster ghosts get the speed boosts they need.

IMPORTANT Instructions:

- Create separate threads for Pac-Man and each ghost to enable independent movement.
- Ensure all threads should start correctly, handling any errors during thread execution.
- Implement concurrent execution allowing simultaneous movement of Pac-Man and ghosts without lag or jitter.
- You are NOT allowed to use any explicit waiting mechanism (e.g. Thread – join or similar) to achieve concurrency/synchronization. You have to use proper synchronization primitives such as semaphores and mutexes to achieve any necessary waits.
- Choose Synchronization solutions Wisely: As each scenario is mapped to different classic problems in concurrent programming, it's critical to select the most appropriate synchronization solution for each challenge to ensure optimal game performance and fairness among game entities.

National University of Computer and Emerging Sciences

FAST School of Computing **Spring 2024** **Islamabad Campus**



Instructions for setting up GLUT Library with C in Linux

Step 1: Install the Required Packages

First, you need to install the necessary development libraries for OpenGL and GLUT. Open a terminal and run the following commands:

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install freeglut3 freeglut3-dev
```

This will install the GCC compiler, make tool, and freeglut libraries, which are an open-source alternative to the original GLUT library.

Step 2: Verify Installation

To ensure that the libraries are installed, you can check the installation paths:

```
dpkg -L freeglut3-dev
```

This command lists the directories where the headers and libraries are installed, which are typically found in /usr/include/GL for headers and /usr/lib/x86_64-linux-gnu for libraries.

Step 3: Write a Basic GLUT Program

Create a new file named main.c and open it in your favorite text editor. Input the following code, which creates a basic GLUT program where an object can be moved using the arrow keys:

```c
#include <GL/glut.h>

// Initial square position and size
float x = 50.0f;
float y = 50.0f;
float side = 50.0f;

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

    // Draw square
    glBegin(GL_QUADS);
        glVertex2f(x, y);
```

```
        glVertex2f(x + side, y);
        glVertex2f(x + side, y + side);
        glVertex2f(x, y + side);
    glEnd();

    glutSwapBuffers();
}

void keyboard(int key, int xx, int yy) {
    switch (key) {
        case GLUT_KEY_RIGHT:
            x += 10;
            break;
        case GLUT_KEY_LEFT:
            x -= 10;
            break;
        case GLUT_KEY_UP:
            y -= 10;
            break;
        case GLUT_KEY_DOWN:
            y += 10;
            break;
    }
    glutPostRedisplay();
}

void initOpenGL() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 300, 300, 0);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(300, 300);
```

```
    glutInitWindowPosition(100, 100);
    glutCreateWindow("OpenGL Setup Test");
    initOpenGL();
    glutDisplayFunc(display);
    glutSpecialFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

Step 4: Compile the Program

To compile the program, use the following command in the terminal:

```
gcc main.c -o main -lGL -lGLU -lglut
```

This command compiles the main.c file and links it against the OpenGL, GLU, and GLUT libraries.

Step 5: Run the Program

After compiling, you can run the program by typing:

```
./main
```

This will open a window with a square that can be moved around using the arrow keys.

Important Note:

In case any group is not able to or does not want to implement a game with a proper GUI using the GLUT/SFML library, they can instead make a terminal-based game with all the features and clearly showcase all the OS concepts that are required in the project.
However, this will come with a penalty of 30 marks from the overall score of 160. These students will also no longer be eligible for the bonus 20 marks for game aesthetics as well, even if they complete rest of the components. This will result in the maximum score attainable being 130. Also its would be hard to witness the synchronization between different threads happening with proper graphical interface. Therefore, it will be sole responsibily of the students to showcase that the synchronization happening between different threads stays recognizable. In short, graphical Pac-Man is highly recommended.

National University of Computer and Emerging Sciences
FAST School of Computing **Spring 2024** **Islamabad Campus**

RUBRICS
SELF EVALUATION SHEET

Evaluation Criteria	Total Marks (160) Bonus Marks(30)	**Self evaluated Marks by students**	(Do NOT Fill Marks) Obtained Member 1 Marks	(Do NOT Fill Marks) Obtained Member 1 Marks	
			Name:	Name:	
			Roll #:	Roll #:	
Background and Foreground Threads creation (45 Marks)					
Game Engine Thread: - Main game loop execution (5 marks) - Input handling (5 marks) - Game state update (5 marks) - Rendering (5 marks)	20				
User Interface Thread: - UI component management (5 marks) - Input event handling (5 marks) - Game state display (5 marks)	15				
Ghost Controller Thread(s): - Individual ghost controllers (5 marks) - Ghost algorithm execution (5 marks)	10				
Design and Implementation (35 Marks)					
Game Board Initialization: Game board layout with clear paths, walls, and pellet positions.	10				
Implementation of Game and Eating Mechanics: Assess the accuracy and completeness of basic game	15				

mechanics, including Pac-Man and ghost movements, as well as pellet consumption.					
Implementation of Lives system: allowing the player to have multiple attempts at completing the game.	10				
Synchronization (80 Marks)					
Synchronization Scenario 1 implementation: Rate the handling and correct implementation of Synchronization scenario 1, ensuring its addressed correctly and effectively.	20				
Synchronization Scenario 2 implementation: Rate the handling and correct implementation of Synchronization scenario 2, ensuring its addressed correctly and effectively	20				
Synchronization Scenario 3 implementation: Rate the handling and correct implementation of Synchronization scenario 3, ensuring its addressed correctly and effectively	20				
Synchronization Scenario 4 implementation: Rate the handling and correct implementation of Synchronization scenario 4, ensuring its addressed correctly and effectively	20				
Bonus (30) - Eligibility Criteria (Only in consideration if all other modules are fully complete)					
Github: Bonus marks for proficient GitHub utilization, including clear commit history and messages.	10				
Aesthetics: Bonus Marks for making the base PACMAN game aesthetically pleasing with good visuals.	20				