UNIVERSITI MALAYA

WIF3005 Alternative Assessment

Question 1

LECTURER :

DR. NUR NASUHA BINTI MOHD DAUD

Prepared by :

CHIA PEI XIN (U2102773/1)

Component 1: Game Loop Engine

1a)

Clear explanations

The Game Loop is the core engine that drives the flow of any interactive game or real-time application. It ensures a seamless, immersive, and responsive experience for users by consistently managing the timing and execution of critical functions in a structured manner.

Functionality

1. **Initialization**: Before the loop begins, the game sets up its state, preparing elements like player positions, game objects, or initial settings. This stage ensures that everything is ready for the gameplay to start.

2. **Update**: During this phase, the game continuously calculates and adjusts its internal variables. It handles changes such as object movements, collision detections, and updates to player scores. This ensures that the game logic reflects the current state of play.

3. **Render**: After updating the game state, the render function redraws the screen to visually represent these changes to the player. This keeps the game visually accurate and engaging.

4. **Timing Control**: The Game Loop also manages the frame rate, ensuring the game runs smoothly without causing lag or overloading system resources. It controls how often the update and render functions are executed, maintaining a consistent performance regardless of the device.

Purpose

1. The Game Loop guarantees synchronization between game logic and visual representation, offering an uninterrupted, fluid gameplay experience.

2. It provides precise control over updates and rendering, ensuring accuracy and efficiency in real-time interactions.

Reusability

This component is highly reusable because it provides a universal structure required for real-time applications. Beyond games, it is vital for simulations, data visualizations, and dashboards—any scenario where the system needs to update and display information in real-time. Its modular nature allows developers to plug in specific logic for different contexts, making it a foundational tool for interactive application development.

1b)

Snapshots

```
//-----------------------------------------------------------------------
// GAME LOOP
//-----------------------------------------------------------------------

function run() {

  showStats(); // initialize FPS counter
  addEvents(); // attach keydown and resize events

  var last = now = timestamp();
  function frame() {
    now = timestamp();
    update(Math.min(1, (now - last) / 1000.0)); // using requestAnimationFrame have to be able to handle large delta's caused when
    draw();
    stats.update();
    last = now;
    requestAnimationFrame(frame, canvas);
  }

  resize(); // setup all our sizing information
  reset();  // reset the per-game variables
  frame();  // start the first frame

}
```

The game loop is the core mechanism that keeps the game running by repeatedly updating the game state and rendering it on the screen. It ensures smooth gameplay by managing the flow of game logic and visuals.

The **run()** function serves as the starting point of the game. It initializes some key components like the FPS (frames per second) counter using showStats(), which helps track performance, and sets up event listeners through addEvents() for actions like key presses and window resizing. This ensures that the game is interactive and responsive. Once the setup is done, the function starts the game loop by calling a nested frame() function.

The **frame()** function is the heart of the game loop. Within the frame() function, the update() function is called to handle all game-related logic. This includes tasks such as moving Tetris blocks as they fall, checking for collisions (e.g., when a block lands or collides with another block), clearing rows when they are completed, and keeping track of the player's score. The update() function ensures that the game state evolves dynamically in response to both user inputs and elapsed time. After the game logic has been updated, the draw() function is executed to visually reflect these changes on the screen. For example, if a Tetris block moves down one row, the draw() function updates the game canvas to show the block's new position. This real-time rendering ensures that players see the immediate results of their actions and the game's progression. To keep everything running smoothly, the browser's requestAnimationFrame is used to synchronize the game updates with the display's refresh rate, avoiding any unnecessary lag or jitter.

The snapshot also highlights the importance of adaptability and preparation. The **resize()** function ensures that the game's canvas adjusts dynamically to fit different screen sizes. It recalculates the size of each Tetris block and resets the layout whenever the window is resized. This ensures a consistent experience for players, regardless of the device or window size. Similarly, the **reset()** function initializes all game-specific variables, preparing everything for the start of a new session.

Overall, this code snippet captures the essential structure of a game loop, showing how different components like game logic, rendering, input handling, and timing come together. It's a modular and efficient design that ensures smooth gameplay and a responsive user experience, making it a fundamental example of how a Game Loop Engine operates in games.

1c)

Usage Examples of the Game Loop in Real-World Scenarios

1. **Interactive Simulations**
    - Example: Simulating the motion of a ball bouncing off walls.
    - The update() function regularly calculates the ball's position based on physics rules.
    - The draw() function displays the updated position on the screen, ensuring smooth and accurate visuals.
    - The resize() function ensures the simulation adapts to screen size changes, maintaining consistency across devices.

2. **Real-Time Data Dashboards**
    - Example: Visualizing stock market prices or sensor data.
    - The update() function fetches new data from a server or data source.
    - The draw() function renders this data in real-time charts or graphs.
    - The resize() function adjusts the layout dynamically if the user resizes the browser window.

3. **Animation and Graphics Rendering**
    - Example: Animating a character walking or rotating a 3D object.
    - The update() function controls the movement or transformation of objects.
    - The draw() function ensures smooth rendering of animations on the screen.
    - The resize() function resizes the canvas or view, maintaining quality across resolutions.

4. **Interactive User Interfaces (UIs)**
    - Example: Sliding menus or animated pop-up windows.
    - The update() function processes user actions, such as clicks or scrolling.
    - The draw() function animates the transitions for smooth UI interactions.
    - The resize() function ensures the UI remains responsive and fits all screen sizes.

5. **Game Engines Beyond Tetris**
    - Example: Platformers, strategy games, or first-person shooters.

- The update() function handles game logic, such as character movement and collision detection.
- The draw() function renders the game world and updates the visuals in real-time.
- The resize() function ensures the game scales properly across various screen sizes.

6. **Virtual Reality (VR) Applications**
    - Example: Managing real-time updates in immersive VR environments.
    - The update() function tracks the user's movements and interactions with the virtual world.
    - The draw() function renders the immersive environment for a seamless VR experience.
    - The resize() function adjusts the environment's scale for different headsets or screen resolutions.

7. **Automation and Robotics**
    - Example: Handling continuous sensor data and actuator updates.
    - The update() function processes sensor input, such as motion or distance data.
    - The draw() function visualizes this data on a screen for operators.
    - While the resize() function may not always be needed, the loop structure ensures efficient and continuous operation.

8. **AI Training and Machine Learning Projects**
    - Example: Visualizing the performance of a machine learning model.
    - The update() function calculates and tracks metrics such as accuracy or loss over time.
    - The draw() function displays real-time graphs or charts reflecting the model's progress.
    - The resize() function ensures the visualization adjusts smoothly to different screen sizes.

The game loop's modular structure of continuous update(), draw(), and resize() functions makes it versatile and applicable across various domains. Its ability to manage real-time updates and responsiveness ensures smooth performance in gaming, simulations, data visualizations, robotics, VR/AR applications, and more. This adaptability makes it an invaluable tool for creating dynamic and interactive experiences.

Component 2: Input Handling Module

1a)

Clear explanations

The Input Handling Module is a robust and flexible system designed to detect, process, and map user inputs seamlessly across various devices and platforms. It acts as a bridge between the user and the system, enabling real-time responsiveness and intuitive control. This module facilitates dynamic interactions, empowering users to manipulate game characters, trigger actions, and engage with the game world in an immersive and interactive manner.

Functionality

1. **Event Listener**: The game listens for specific events, like a key press. Event listeners are optimized for performance and low-latency response to ensure smooth interaction.

2. **Input Mapping**: Allows developers to define and customize mappings between user inputs and in-game actions. For example, mapping the ArrowUp key to the “move forward action.

3. **Key Codes**: When a key is pressed, the system captures its corresponding key code (e.g., ArrowLeft, ArrowRight, Space, or Esc) to identify which key triggered the action to ensure accurate responses.

4. **Real-time Input Processing**: Once the game receives the input, it processes it by triggering actions like moving in a direction, starting a new game, or ending the game.

Purpose

1. The Input Handling Module ensures accurate and responsive interaction between the user and the system.

2. It supports real-time input recognition, providing a smooth and intuitive user experience.

3. Offers developers flexibility to customize input mappings and extend functionality to suit various gameplay scenarios.

Reusability

The Input Handling Module is an essential component for a wide range of interactive systems, including games, simulations, virtual reality applications, and data visualizations. Its modular design enables seamless integration and customization, allowing developers to adapt it to diverse platforms and devices effortlessly. By providing real-time responsiveness and intuitive control, the module ensures accurate input recognition and processing, making it indispensable for creating engaging and immersive user experiences across various interactive systems.

1b)

Snapshots

The first aspect to capture is the **core event handling code**. For example, the keydown event listener plays a crucial role in detecting user input. This forms the foundation for how a component interacts with the user through keyboard events.

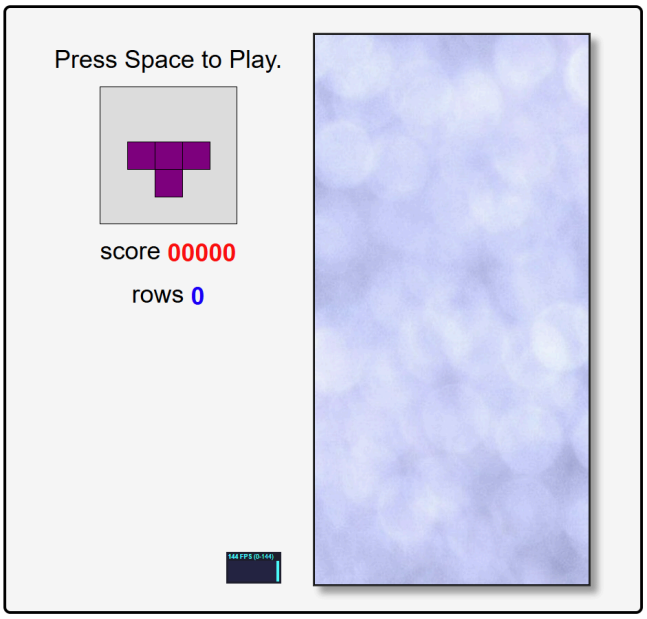
```
function addEvents() {
  document.addEventListener('keydown', keydown, false);
  window.addEventListener('resize', resize, false);
}
```

Following this, the **keydown function** processes specific key presses such as LEFT, RIGHT, UP, and DOWN. For example, the logic to move a piece left when the LEFT arrow key is pressed showcases how the interaction is handled programmatically. This functionality controls the relationship between the user's input and the game's behavior, making the component interactive and dynamic.

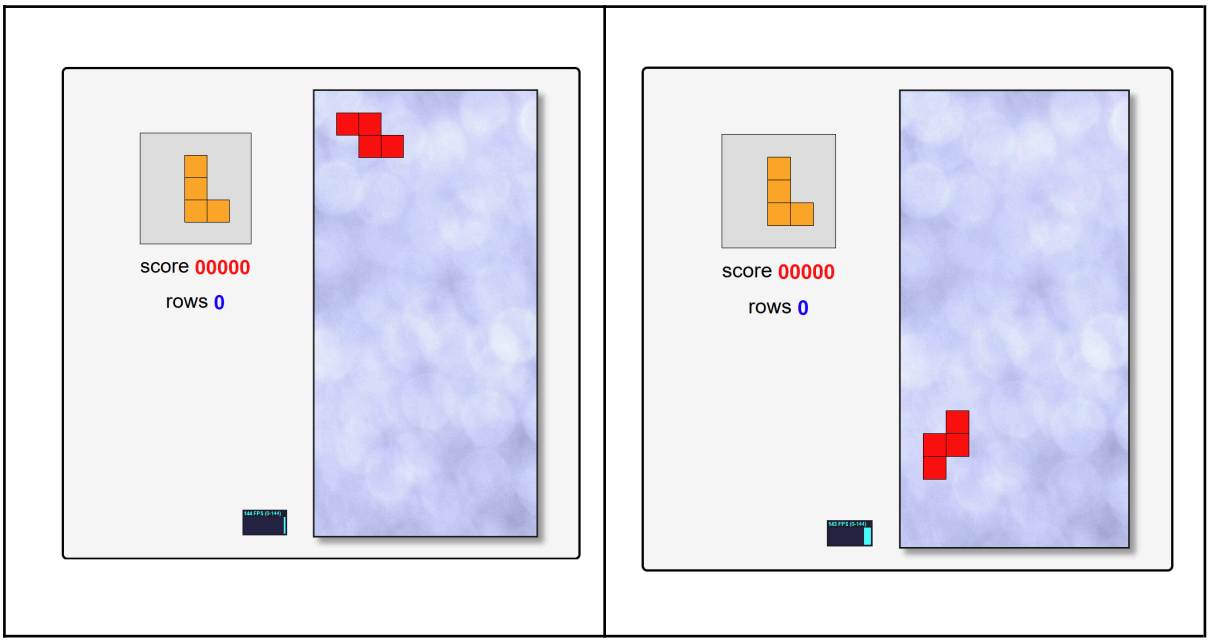
```
function keydown(ev) {
  var handled = false;
  if (playing) {
    switch(ev.keyCode) {
      case KEY.LEFT:   actions.push(DIR.LEFT);  handled = true; break;
      case KEY.RIGHT:  actions.push(DIR.RIGHT); handled = true; break;
      case KEY.UP:     actions.push(DIR.UP);    handled = true; break;
      case KEY.DOWN:   actions.push(DIR.DOWN);  handled = true; break;
      case KEY.ESC:    lose();                  handled = true; break;
    }
  }
  else if (ev.keyCode == KEY.SPACE) {
    play();
    handled = true;
  }
  if (handled)
    ev.preventDefault(); // prevent arrow keys from scrolling the page (supported in IE9+ and all other browsers)
}
```

Examples Across Games

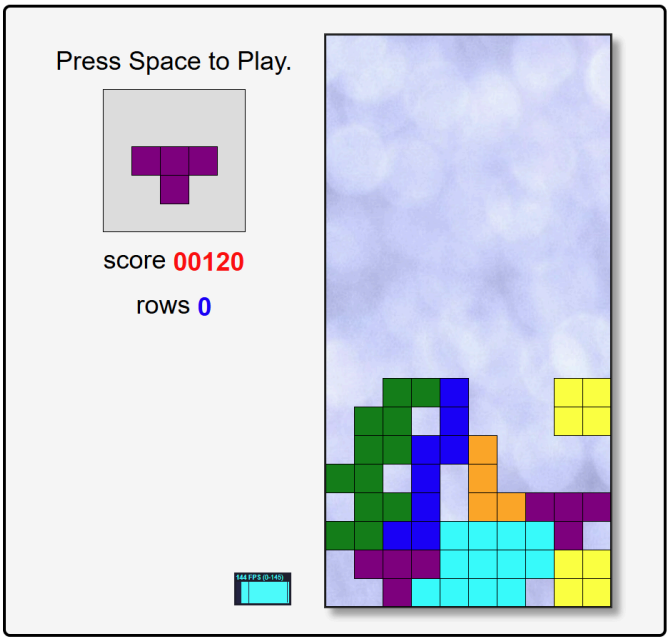
Tetris Game:



Initial State: This is the initial state of the game. When the user presses the SPACE bar, the game transitions to the active state, starting the gameplay.

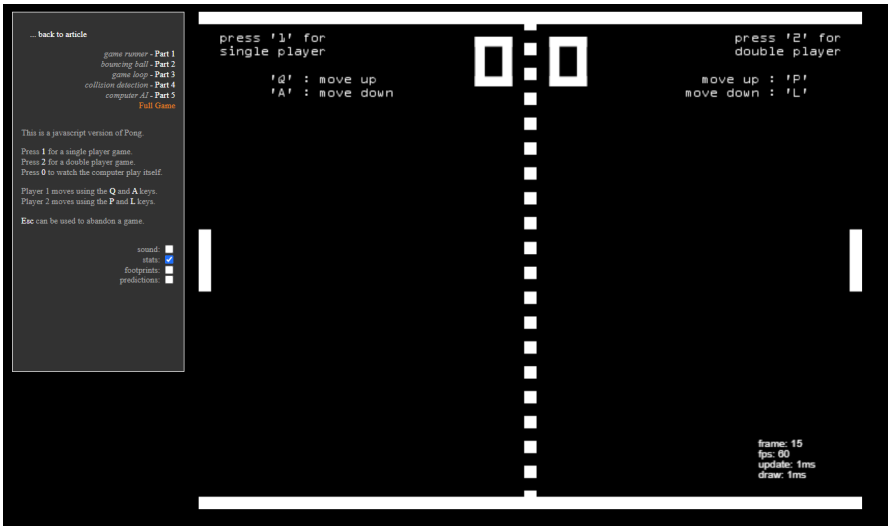


Key Interaction: Pressing the UP key rotates a piece, and this interaction should be visually represented. For instance, a screenshot of the rotated piece demonstrates the effectiveness of the key press.

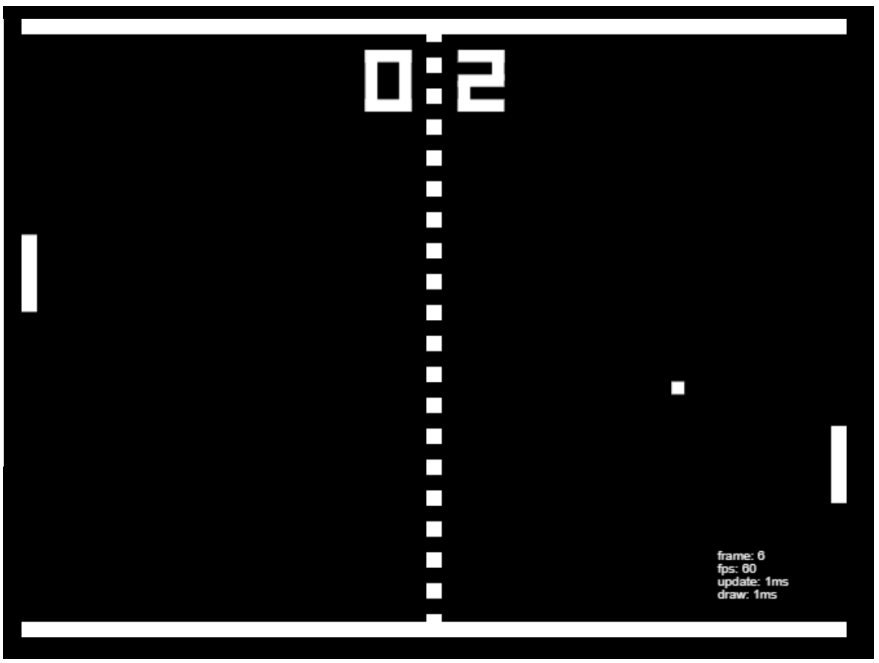


Game Over State: When the ESC key is pressed, the lose() function ends the game. Capturing this state highlights the end-to-end interaction flow.

Pong Game:

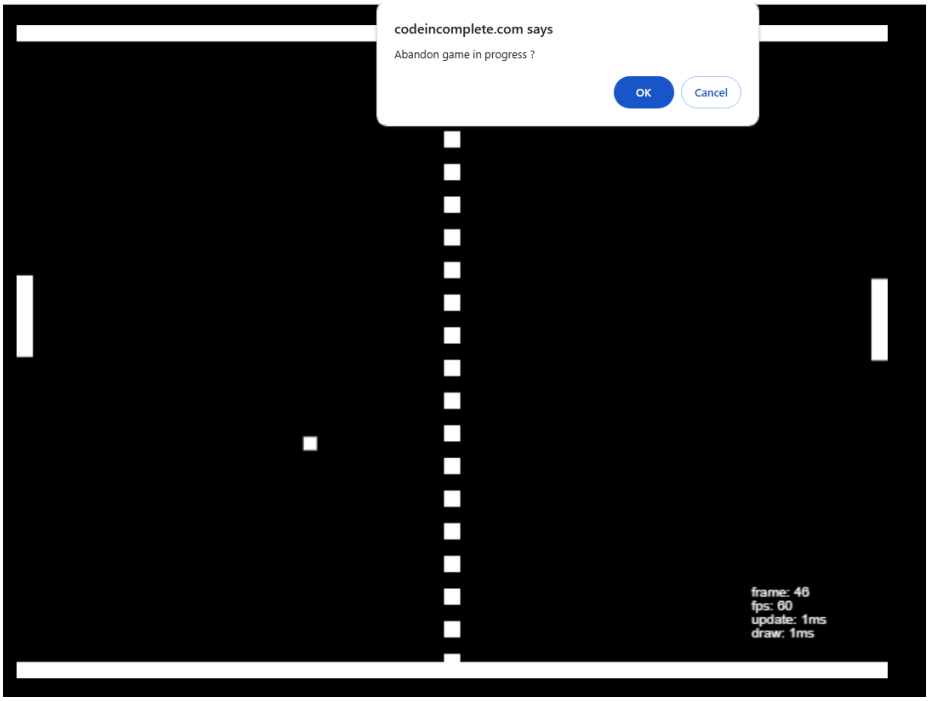


Initial State: Display the game setup where the user selects either 1 (for single-player mode) or 2 (for multiplayer mode). The game begins upon making a selection.



Key Interaction: Illustrate the movement of paddles controlled by players. For example:

- Player 1 uses Q and A keys.
- Player 2 uses P and L keys. A screenshot showing the paddles moving in response to these key presses demonstrates the functionality.



Game End State: Pressing the ESC key stops the game, triggering the lose() function. A confirmation window should pop up, and this should be captured to show how the interaction concludes.

In racer game,

Key Interaction: Show the vehicle's movement controlled by the UP, DOWN, LEFT, and RIGHT arrow keys. For instance:

- Pressing UP moves the car forward.
- Pressing LEFT turns the car. Screenshots of these movements visually demonstrate the real-time impact of user input on the game.

1c)

Usage Examples of the Game Loop in Real-World Scenarios

1. **Web Development**
    - **Example**: User interaction with forms.
    - Input handling processes user data as they type details like name or email.
    - The system validates the data in real time, providing feedback such as error messages or confirmation of successful submissions.
    - This ensures a smooth user experience and that accurate data is sent to the server.

2. **Virtual Reality (VR)**
    - **Example**: Interacting with a virtual environment using controllers or gestures.
    - Specialized controllers, sensors, or gestures capture user movements and translate them into actions in real-time.
    - Input handling adjusts the virtual perspective or triggers animations, ensuring a fluid and immersive experience.
    - This responsiveness is critical for maintaining the natural flow of VR interactions.

3. **Robotics**
    - **Example**: Robots reacting to sensors and user commands.
    - Input handling processes data from joysticks, motion detectors, or environmental sensors.
    - Robots perform tasks like movement, navigation, or object manipulation based on the input.
    - For autonomous robots, input handling interprets environmental feedback to make decisions in real time.

4. **Interactive User Interfaces (UIs)**
    - **Example**: Clicking a button, dragging a slider, or using dropdown menus.
    - Input handling captures user actions and updates the interface dynamically.
    - Actions like changing button states, loading new content, or transitioning pages are enabled by responsive input handling.
    - This ensures the interface feels intuitive and enhances the overall user experience.

Input handling is the backbone of all interactive systems, seamlessly transforming user actions into meaningful system responses. Whether it's validating form data in web development, creating immersive interactions in VR, enabling robots to respond to commands, or making UIs responsive to user inputs, input handling ensures real-time feedback and an engaging experience across a wide range of technologies.