Nama : Irfan Hidayatulah Putra

NIM : 20220140142

Kelas : B

```
let player;
let bullets = [];
let enemies = [];
let spawnTimer = 0;
let score = 0;
let lives = 3;
let gameState = 'menu'; // 'menu', 'playing', 'gameover'

// Sound variables
let shootSound;      // external mp3 for shooting
let gameOverSound;   // external wav for game over

// Auto-fire
let autoFire = false;
let autoFireBtn; // p5 button element

// Difficulty
let difficulties = {
  'Easy':   { lives: 5,  spawnRange: [50, 100], enemySpeedMul: 0.8, cooldownMax: 16 },
  'Normal': { lives: 3,  spawnRange: [30, 80],  enemySpeedMul: 1.0, cooldownMax: 12 },
  'Hard':   { lives: 2,  spawnRange: [15, 50],  enemySpeedMul: 1.4, cooldownMax: 8  }
};
let difficultyNames = ['Easy', 'Normal', 'Hard'];
let selectedDifficulty = 'Normal';

// menu box layout (calculated in setup)
let diffBox = {
  x: 0, y: 0, w: 140, h: 48, spacing: 20
};

function preload() {
  soundFormats('mp3', 'wav');
  shootSound = loadSound('mixkit-game-gun-shot-1662.mp3',
    () => {},
    (err) => { console.warn('Gagal memuat suara tembakan:', err); }
  );
  gameOverSound = loadSound('mixkit-explosion-hit-1704.wav',
```

```
    () => {},
    (err) => { console.warn('Gagal memuat suara gameover:', err); }
  );
}

function setup() {
  // create canvas and UI
  let cnv = createCanvas(800, 600);
  cnv.parent(document.body);
  textAlign(CENTER, CENTER);

  // position difficulty boxes horizontally centered
  let totalW = difficultyNames.length * diffBox.w + (difficultyNames.length - 1) * diffBox.spacing;
  diffBox.x = width / 2 - totalW / 2;
  diffBox.y = height / 2 + 10;

  player = new Player(width / 2, height - 40);
  spawnTimer = 20;

  // Create Auto-Fire toggle button
  autoFireBtn = createButton('Auto-Fire: OFF');
  autoFireBtn.position(width - 130, 8);
  autoFireBtn.style('padding', '6px 8px');
  autoFireBtn.style('font-family', 'sans-serif');
  autoFireBtn.mousePressed(toggleAutoFire);
  autoFireBtn.attribute('aria-label', 'Toggle auto fire');

  // Optional: show hint about key shortcut
  let hint = createP("Tekan 'F' untuk toggle Auto-Fire | Gunakan ← → untuk pilih difficulty, ENTER untuk mulai");
  hint.position(8, height + 8);
  hint.style('margin', '0px');
  hint.style('font-family', 'sans-serif');
  hint.style('font-size', '12px');

  // apply initial difficulty (so UI shows correct values)
  applyDifficultySettings(selectedDifficulty);
}

function draw() {
  background(30);

  if (gameState === 'menu') {
```

```
  drawMenu();
  return;
}

if (gameState === 'playing') {
  // --- INPUT handled per-frame so movement + shooting can occur simultaneously ---
  handleInput();

  // If autoFire is ON, request shoot every frame (player.shoot() respects cooldown)
  if (autoFire) {
    player.shoot();
  }

  // spawn enemies (use difficulty spawn range, scale with score)
  if (spawnTimer <= 0) {
    enemies.push(new Enemy(random(20, width - 20), -20));
    spawnTimer = getNextSpawnTimer();
  } else {
    spawnTimer--;
  }

  // update player
  player.update();
  player.show();

  // update bullets
  for (let i = bullets.length - 1; i >= 0; i--) {
    bullets[i].update();
    bullets[i].show();
    if (bullets[i].offscreen()) bullets.splice(i, 1);
  }

  // update enemies
  for (let i = enemies.length - 1; i >= 0; i--) {
    enemies[i].update();
    enemies[i].show();

    // enemy hits bottom -> lose life
    if (enemies[i].y - enemies[i].h/2 > height) {
      enemies.splice(i, 1);
      lives--;
      if (lives <= 0) {
```

```
          triggerGameOver();
        }
        continue;
      }

      // enemy collides with player
      if (enemies[i].hitsPlayer(player)) {
        enemies.splice(i, 1);
        lives--;
        if (lives <= 0) {
          triggerGameOver();
        }
        continue;
      }

      // bullet collisions
      for (let j = bullets.length - 1; j >= 0; j--) {
        if (enemies[i] && bullets[j] && enemies[i].hitsBullet(bullets[j])) {
          playExplosion(enemies[i].x, enemies[i].y);
          enemies.splice(i, 1);
          bullets.splice(j, 1);
          score += 10;
          break;
        }
      }
    }

    // HUD
    drawHUD();
  } else if (gameState === 'gameover') {
    drawGameOver();
  }
}

function getNextSpawnTimer() {
  // calculates next spawn timer based on selected difficulty and score pressure
  let cfg = difficulties[selectedDifficulty];
  let minV = cfg.spawnRange[0];
  let maxV = cfg.spawnRange[1];
  // scale down spawn times slowly as score increases
  let scaleDown = floor(score / 50);
  let minAdj = max(6, minV - scaleDown);
```

```
  let maxAdj = max(minAdj + 6, maxV - floor(score / 30));
  return int(random(minAdj, maxAdj));
}

function applyDifficultySettings(name) {
  // apply difficulty settings to global gameplay variables
  let cfg = difficulties[name];
  if (!cfg) cfg = difficulties['Normal'];
  lives = cfg.lives;
  // set player's cooldown max (so auto-fire / manual respect difficulty)
  if (player) player.cooldownMax = cfg.cooldownMax;
  // spawnTimer initial
  spawnTimer = getNextSpawnTimer();
}

function handleInput() {
  // movement
  if (keyIsDown(LEFT_ARROW) || keyIsDown(65)) { // LEFT or A
    player.setDir(-1);
  } else if (keyIsDown(RIGHT_ARROW) || keyIsDown(68)) { // RIGHT or D
    player.setDir(1);
  } else {
    player.setDir(0);
  }

  // shooting: tahan space untuk menembak sesuai cooldown
  if (keyIsDown(32)) { // SPACE
    player.shoot();
  }
}

function drawMenu() {
  fill(255);
  textSize(48);
  text('Space Shoot', width / 2, height / 2 - 100);

  textSize(16);
  fill(200);
  text('Pilih tingkat kesulitan: (klik kotak / gunakan ← → lalu ENTER)', width / 2, height / 2 - 60);

  // draw difficulty boxes
  for (let i = 0; i < difficultyNames.length; i++) {
```

```
    let name = difficultyNames[i];
    let x = diffBox.x + i * (diffBox.w + diffBox.spacing);
    let y = diffBox.y;
    // box background
    if (name === selectedDifficulty) {
      fill(255, 204, 0);
      stroke(255);
      strokeWeight(2);
    } else {
      fill(60);
      noStroke();
    }
    rectMode(CORNER);
    rect(x, y, diffBox.w, diffBox.h, 8);

    // label
    noStroke();
    fill(0);
    if (name === selectedDifficulty) {
      fill(20);
    } else {
      fill(220);
    }
    textSize(18);
    text(name, x + diffBox.w / 2, y + diffBox.h / 2 - 6);

    // small details (lives / speed)
    let cfg = difficulties[name];
    textSize(12);
    fill(name === selectedDifficulty ? 20 : 200);
    text(`Lives: ${cfg.lives}`, x + diffBox.w / 2, y + diffBox.h / 2 + 10);
  }

  // show controls hint
  fill(200);
  textSize(14);
  text('A / ←  : kiri  |  D / → : kanan  |  SPACE : tembak', width / 2, diffBox.y + diffBox.h + 36);
  text("Tekan ENTER atau klik area kosong untuk mulai dengan pilihan saat ini.", width / 2, diffBox.y + diffBox.h + 56);
}

function drawHUD() {
  fill(255);
```

```
  textSize(14);
  textAlign(LEFT, TOP);
  text('Score: ' + score, 8, 8);
  text('Lives: ' + lives, 8, 28);

  // show difficulty and auto-fire status top-right
  textAlign(RIGHT, TOP);
  let level = (1 + floor(score / 100));
  text('Level: ' + level, width - 8, 8);

  // small indicator below Level
  textSize(12);
  textAlign(RIGHT, TOP);
  text('Diff: ' + selectedDifficulty, width - 8, 28);
  text('Auto-Fire: ' + (autoFire ? 'ON' : 'OFF'), width - 8, 46);

  textAlign(CENTER, CENTER);
}

function drawGameOver() {
  fill(255, 80, 80);
  textSize(64);
  text('GAME OVER', width / 2, height / 2 - 60);
  textSize(24);
  fill(255);
  text('Score: ' + score, width / 2, height / 2 );
  textSize(16);
  text('Tekan R untuk main lagi', width / 2, height / 2 + 40);
  text('Tekan M untuk kembali ke menu', width / 2, height / 2 + 64);
}

// Start / restart and mouse handling
function keyPressed() {
  if (gameState === 'menu') {
    // change selection by arrow keys
    if (keyCode === LEFT_ARROW) {
      changeSelection(-1);
    } else if (keyCode === RIGHT_ARROW) {
      changeSelection(1);
    } else if (keyCode === ENTER) {
      // start game with selected difficulty
      if (typeof userStartAudio === 'function') userStartAudio();
```

```
      startGame();
    }
  } else if (gameState === 'playing') {
    // F toggles auto-fire
    if (key === 'f' || key === 'F') {
      toggleAutoFire();
    }
    // R handled below for gameover but allow restart quickly
    if (key === 'r' || key === 'R') {
      restartGame();
    }
  } else if (gameState === 'gameover') {
    if (key === 'r' || key === 'R') {
      restartGame();
    }
    if (key === 'm' || key === 'M') {
      // back to menu
      gameState = 'menu';
      // reapply default selected difficulty UI
      applyDifficultySettings(selectedDifficulty);
    }
    // still allow toggling auto-fire from gameover screen
    if (key === 'f' || key === 'F') toggleAutoFire();
  }
}

function changeSelection(dir) {
  let idx = difficultyNames.indexOf(selectedDifficulty);
  idx = (idx + dir + difficultyNames.length) % difficultyNames.length;
  selectedDifficulty = difficultyNames[idx];
  // update UI/preview values
  applyDifficultySettings(selectedDifficulty);
}

function mousePressed() {
  if (gameState === 'menu') {
    // check if clicked on any difficulty box
    if (checkMenuClick(mouseX, mouseY)) {
      // click handled (selection changed) - do not start
      return;
    }
    // otherwise, start the game (click on empty area)
```

```
    if (typeof userStartAudio === 'function') userStartAudio();
    startGame();
  } else if (gameState === 'playing') {
    // mouse click shoots (still works while moving)
    player.shoot();
  } else if (gameState === 'gameover') {
    // click to restart quickly
    restartGame();
  }
}

function checkMenuClick(mx, my) {
  for (let i = 0; i < difficultyNames.length; i++) {
    let x = diffBox.x + i * (diffBox.w + diffBox.spacing);
    let y = diffBox.y;
    if (mx >= x && mx <= x + diffBox.w && my >= y && my <= y + diffBox.h) {
      selectedDifficulty = difficultyNames[i];
      applyDifficultySettings(selectedDifficulty);
      return true;
    }
  }
  return false;
}

function startGame() {
  score = 0;
  bullets = [];
  enemies = [];
  // set initial lives and cooldown based on selected difficulty
  let cfg = difficulties[selectedDifficulty];
  lives = cfg.lives;
  if (player) player.cooldownMax = cfg.cooldownMax;
  spawnTimer = getNextSpawnTimer();
  gameState = 'playing';
}

function restartGame() {
  startGame();
  gameState = 'playing';
}

// trigger game over once and play sound
```

```
function triggerGameOver() {
  if (gameState !== 'gameover') {
    if (typeof userStartAudio === 'function') {
      try { userStartAudio(); } catch (e) {}
    }

    gameState = 'gameover';
    // play game over sound once
    if (gameOverSound && gameOverSound.isLoaded()) {
      try {
        gameOverSound.setVolume(1.0);
        gameOverSound.play();
      } catch (e) {}
    }
  }
}

// Toggle function for Auto-Fire button and shortcut
function toggleAutoFire() {
  autoFire = !autoFire;
  if (autoFireBtn) {
    autoFireBtn.html('Auto-Fire: ' + (autoFire ? 'ON' : 'OFF'));
    // small visual cue
    if (autoFire) {
      autoFireBtn.style('background-color', '#ffdd57');
    } else {
      autoFireBtn.style('background-color', '');
    }
  }
}

// --- Classes ---

class Player {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.w = 48;
    this.h = 18;
    this.speed = 6;
    this.dir = 0;
    this.cooldown = 0; // frames until next shot
```

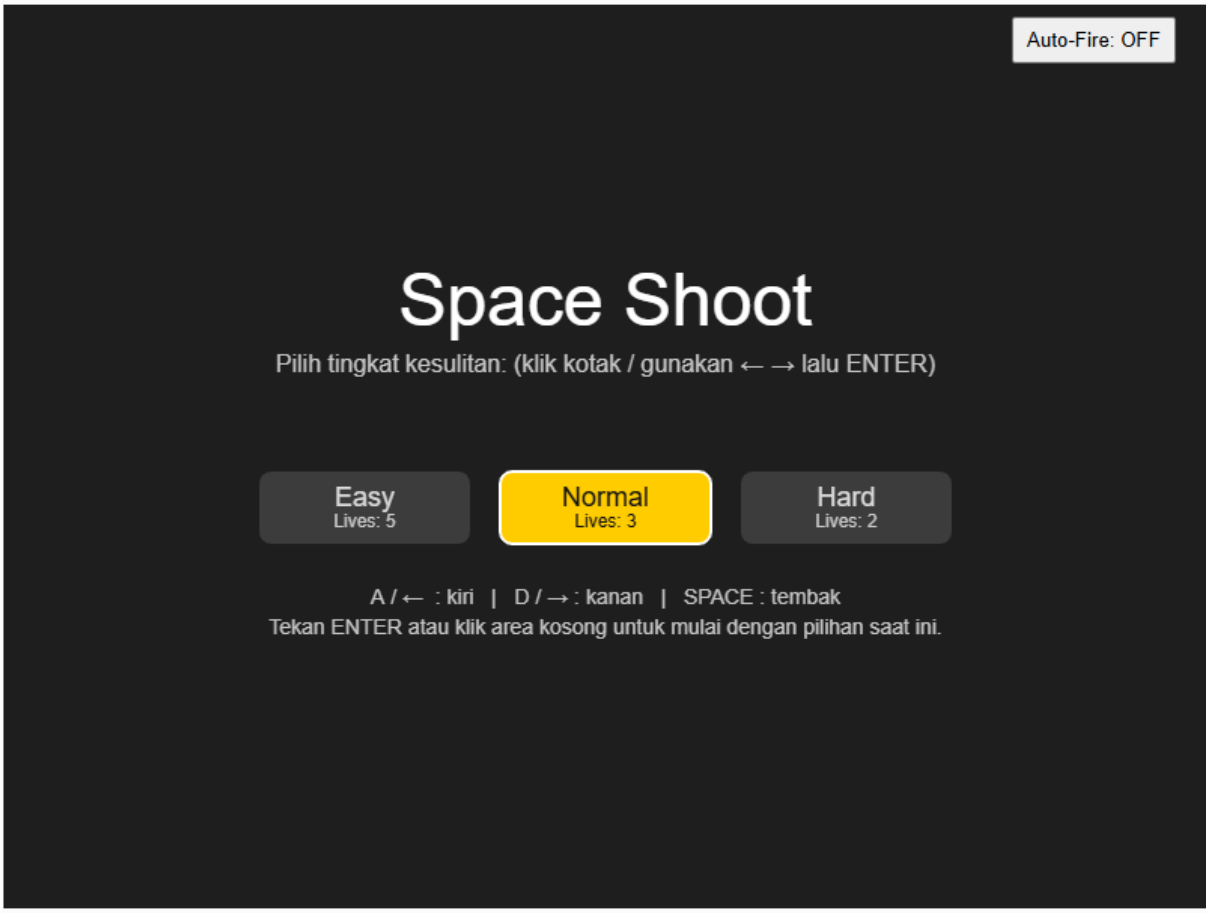
```
    this.cooldownMax = difficulties[selectedDifficulty].cooldownMax; // set by difficulty
  }

  setDir(d) {
    this.dir = d;
  }

  update() {
    this.x += this.dir * this.speed;
    this.x = constrain(this.x, this.w / 2, width - this.w / 2);
    if (this.cooldown > 0) this.cooldown--;
  }

  show() {
    push();
    translate(this.x, this.y);
    noStroke();
    fill(100, 200, 255);
    rectMode(CENTER);
    rect(0, 0, this.w, this.h, 6);
    fill(20, 80, 140);
    triangle(-12, -2, 12, -2, 0, -12);
    pop();
  }

  shoot() {
    if (this.cooldown === 0) {
      bullets.push(new Bullet(this.x, this.y - this.h / 2 - 6));
      if (shootSound && shootSound.isLoaded()) {
        try {
          let rate = random(0.95, 1.05);
          shootSound.rate(rate);
          shootSound.setVolume(0.8);
          shootSound.play();
        } catch (e) {
          try { shootSound.play(); } catch (e2) {}
        }
      }
      this.cooldown = this.cooldownMax;
    }
  }
}
```

```
class Bullet {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.r = 5;
    this.speed = 10;
  }

  update() {
    this.y -= this.speed;
  }

  show() {
    noStroke();
    fill(255, 200, 50);
    circle(this.x, this.y, this.r * 2);
  }

  offscreen() {
    return this.y + this.r < 0;
  }
}

class Enemy {
  constructor(x, y) {
    this.baseX = x;
    this.x = x;
    this.y = y;
    this.w = random(28, 48);
    this.h = this.w * 0.6;
    // speed scales with score and difficulty
    let cfg = difficulties[selectedDifficulty];
    this.speed = random((1 + score / 200) * cfg.enemySpeedMul, (2 + score / 120) * cfg.enemySpeedMul);
    this.osc = random(0.01, 0.05);
    this.angle = random(TWO_PI);
  }

  update() {
    this.y += this.speed;
    this.angle += this.osc;
    this.x = this.baseX + sin(this.angle) * 36;
```

```
  }

  show() {
    push();
    translate(this.x, this.y);
    noStroke();
    fill(200, 100, 120);
    rectMode(CENTER);
    rect(0, 0, this.w, this.h, 8);
    fill(180, 50, 80);
    triangle(-this.w * 0.4, this.h * 0.2, this.w * 0.4, this.h * 0.2, 0, this.h * 0.6);
    pop();
  }

  hitsBullet(b) {
    let dx = abs(b.x - this.x);
    let dy = abs(b.y - this.y);
    if (dx > (this.w / 2 + b.r)) return false;
    if (dy > (this.h / 2 + b.r)) return false;
    return true;
  }

  hitsPlayer(p) {
    return !(p.x + p.w/2 < this.x - this.w/2 ||
             p.x - p.w/2 > this.x + this.w/2 ||
             p.y + p.h/2 < this.y - this.h/2 ||
             p.y - p.h/2 > this.y + this.h/2);
  }
}

// simple explosion synth for enemy destruction
function playExplosion(x, y) {
  try {
    let osc = new p5.Oscillator('sine');
    let env = new p5.Envelope();
    env.setADSR(0.001, 0.05, 0.2, 0.1);
    env.setRange(0.9, 0);
    let baseFreq = random(120, 600);
    osc.freq(baseFreq);
    osc.amp(0);
    osc.start();
    osc.freq(baseFreq * random(0.8, 1.2));
```

```
    env.play(osc);
    setTimeout(() => { try { osc.stop(); } catch (e) {} }, 220);
  } catch (e) {}
}
```

Preview

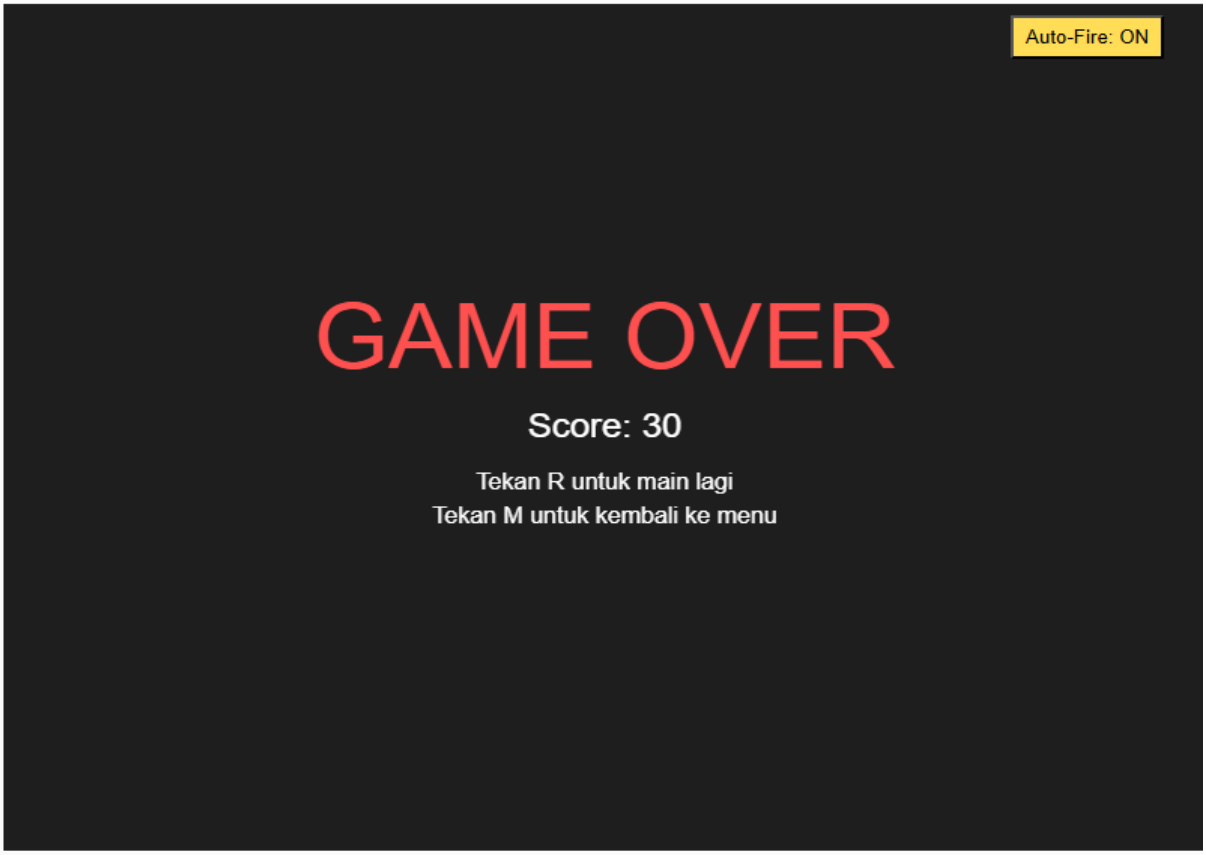


Tekan 'F' untuk toggle Auto-Fire | Gunakan ← → untuk pilih difficulty, ENTER untuk mulai

Preview

Tekan 'F' untuk toggle Auto-Fire | Gunakan ← → untuk pilih difficulty, ENTER untuk mulai

Preview



Tekan 'F' untuk toggle Auto-Fire | Gunakan ← → untuk pilih difficulty, ENTER untuk mulai