SOFE 4850U User interface and Experience Design

Assignment 1

Group Member#1 : Tanish Singla, 100782583

Group Member#2 : Kunal Pandya, 100792272

Group Member#3 : Akshat Gupta, 100813132

Assignment Overview

Link to GitHub Repository:

https://github.com/tanish1409/UX_Assignment_1/tree/main

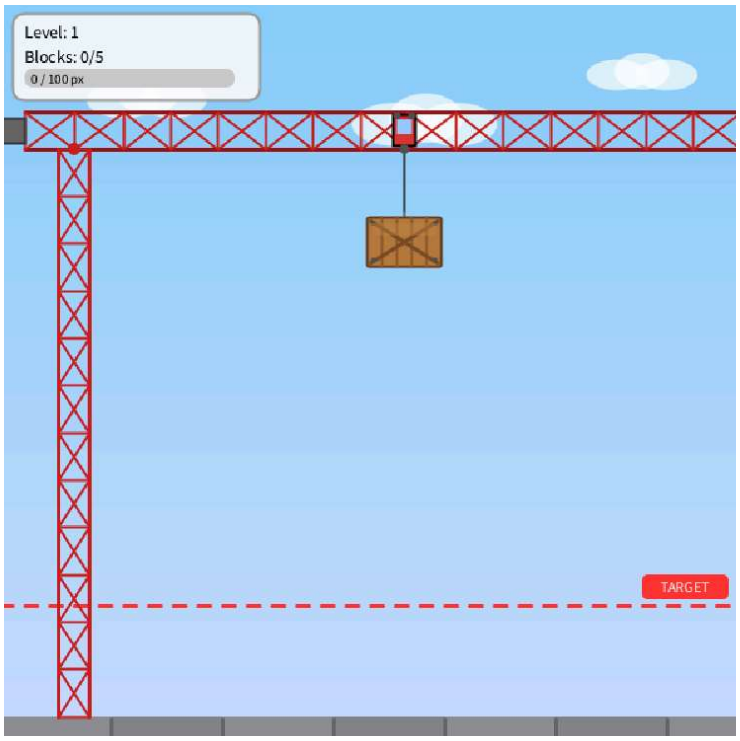
This project is a 2D interactive tower stacking game developed in Processing using the Box2D for Processing physics library. The game features a moving crane that continuously travels back and forth across the top of the screen. The player can drop wooden blocks from the crane to build a tower on the ground. The objective is to stack the blocks above a red target line and maintain tower stability for a few seconds to win the level.

The game includes multiple core UI and gameplay features, including:

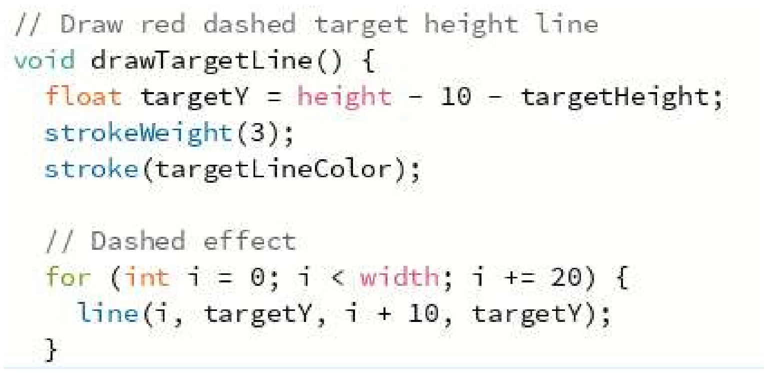
- A crane movement UI with automated trolley motion.
- Real time physics simulation for falling and colliding blocks.
- A target line indicator showing the required height to clear each level.
- A stability timer mechanism that starts when the tower crosses the target height.
- Different game states with separate UIs (Game Over and Level Complete screens).
- A HUD (Heads Up Display) showing the current level, number of blocks dropped, and a progress bar toward the height target.
- Particle effects when blocks are dropped to enhance visual feedback.
- Level progression, where each level increases the difficulty by raising the target height and adding more blocks.
- A clean sky and ground UI with visual details for an appealing interface.

Key Code Snippets and Output

#1 Game start UI

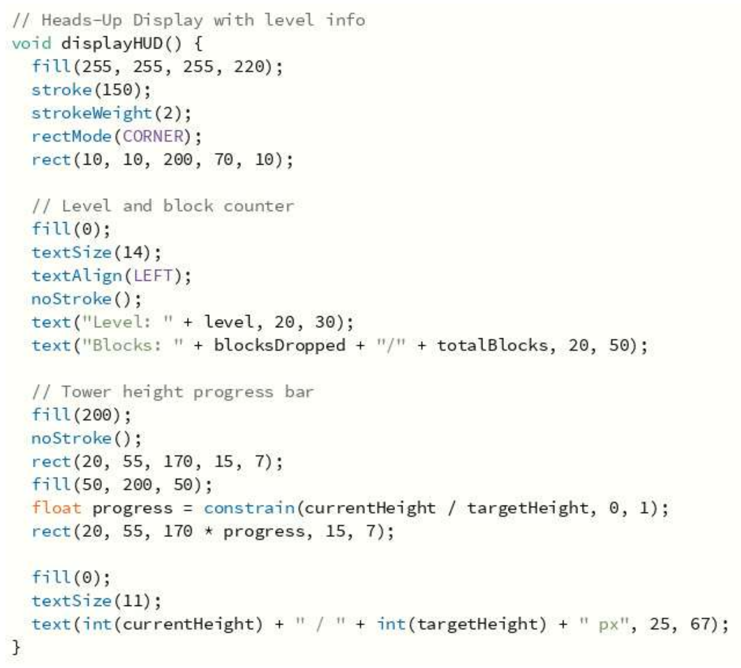


Game start UI



```
// Draw red dashed target height line
void drawTargetLine() {
  float targetY = height - 10 - targetHeight;
  strokeWeight(3);
  stroke(targetLineColor);

  // Dashed effect
  for (int i = 0; i < width; i += 20) {
    line(i, targetY, i + 10, targetY);
  }
```

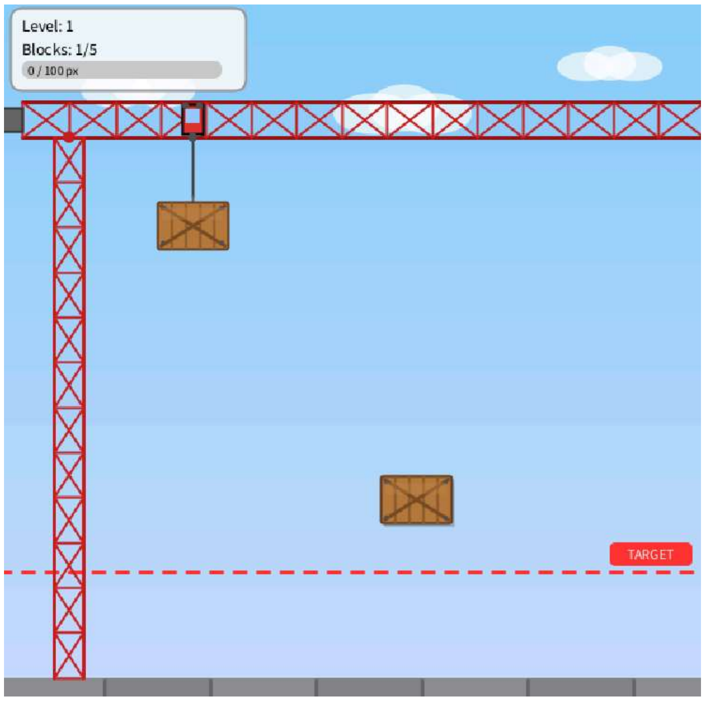
Code snippet to draw target line



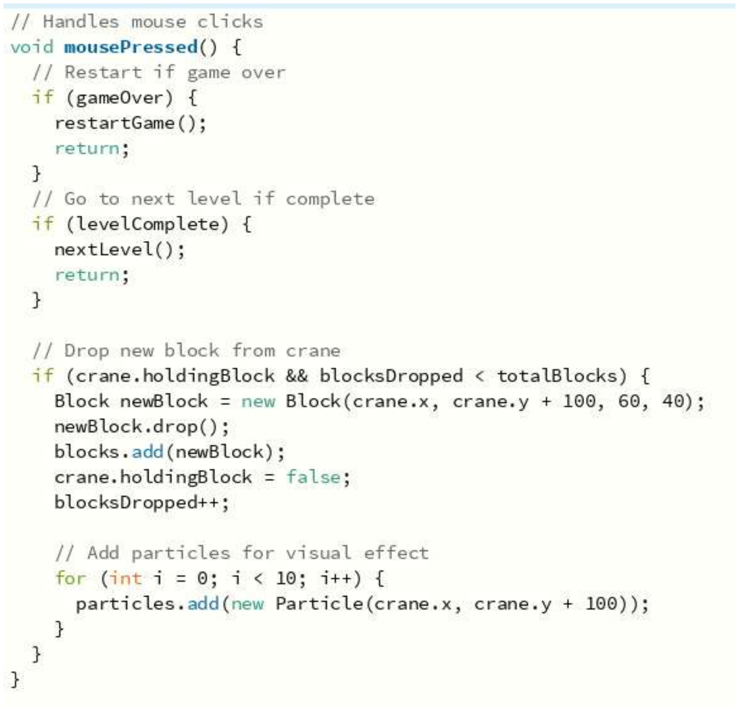
```
// Heads-Up Display with level info
void displayHUD() {
  fill(255, 255, 255, 220);
  stroke(150);
  strokeWeight(2);
  rectMode(CORNER);
  rect(10, 10, 200, 70, 10);

  // Level and block counter
  fill(0);
  textSize(14);
  textAlign(LEFT);
  noStroke();
  text("Level: " + level, 20, 30);
  text("Blocks: " + blocksDropped + "/" + totalBlocks, 20, 50);

  // Tower height progress bar
  fill(200);
  noStroke();
  rect(20, 55, 170, 15, 7);
  fill(50, 200, 50);
  float progress = constrain(currentHeight / targetHeight, 0, 1);
  rect(20, 55, 170 * progress, 15, 7);

  fill(0);
  textSize(11);
  text(int(currentHeight) + " / " + int(targetHeight) + " px", 25, 67);
}
```

Code snippet to display HUD

#2 Crane dropping a block



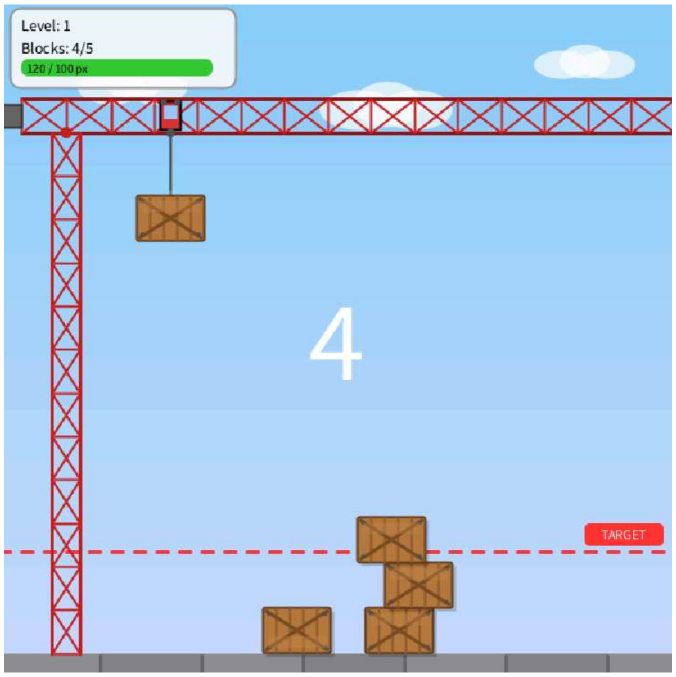
Crane dropping a block on mouseclick



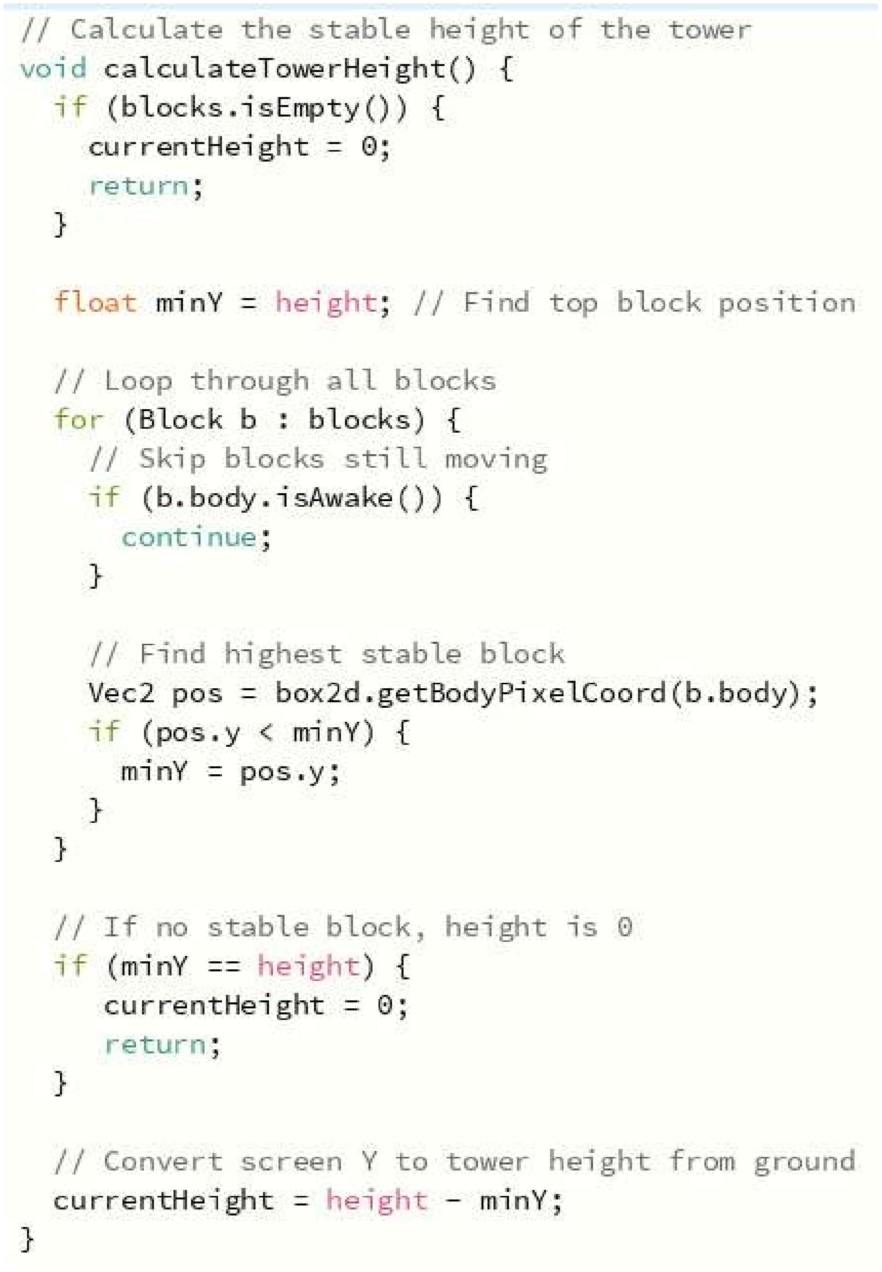
```
// Handles mouse clicks
void mousePressed() {
  // Restart if game over
  if (gameOver) {
    restartGame();
    return;
  }
  // Go to next level if complete
  if (levelComplete) {
    nextLevel();
    return;
  }

  // Drop new block from crane
  if (crane.holdingBlock && blocksDropped < totalBlocks) {
    Block newBlock = new Block(crane.x, crane.y + 100, 60, 40);
    newBlock.drop();
    blocks.add(newBlock);
    crane.holdingBlock = false;
    blocksDropped++;

    // Add particles for visual effect
    for (int i = 0; i < 10; i++) {
      particles.add(new Particle(crane.x, crane.y + 100));
    }
  }
}
```

Code snippet to show how user input triggers block drop

#3 Tower reaches target line and countdown starts

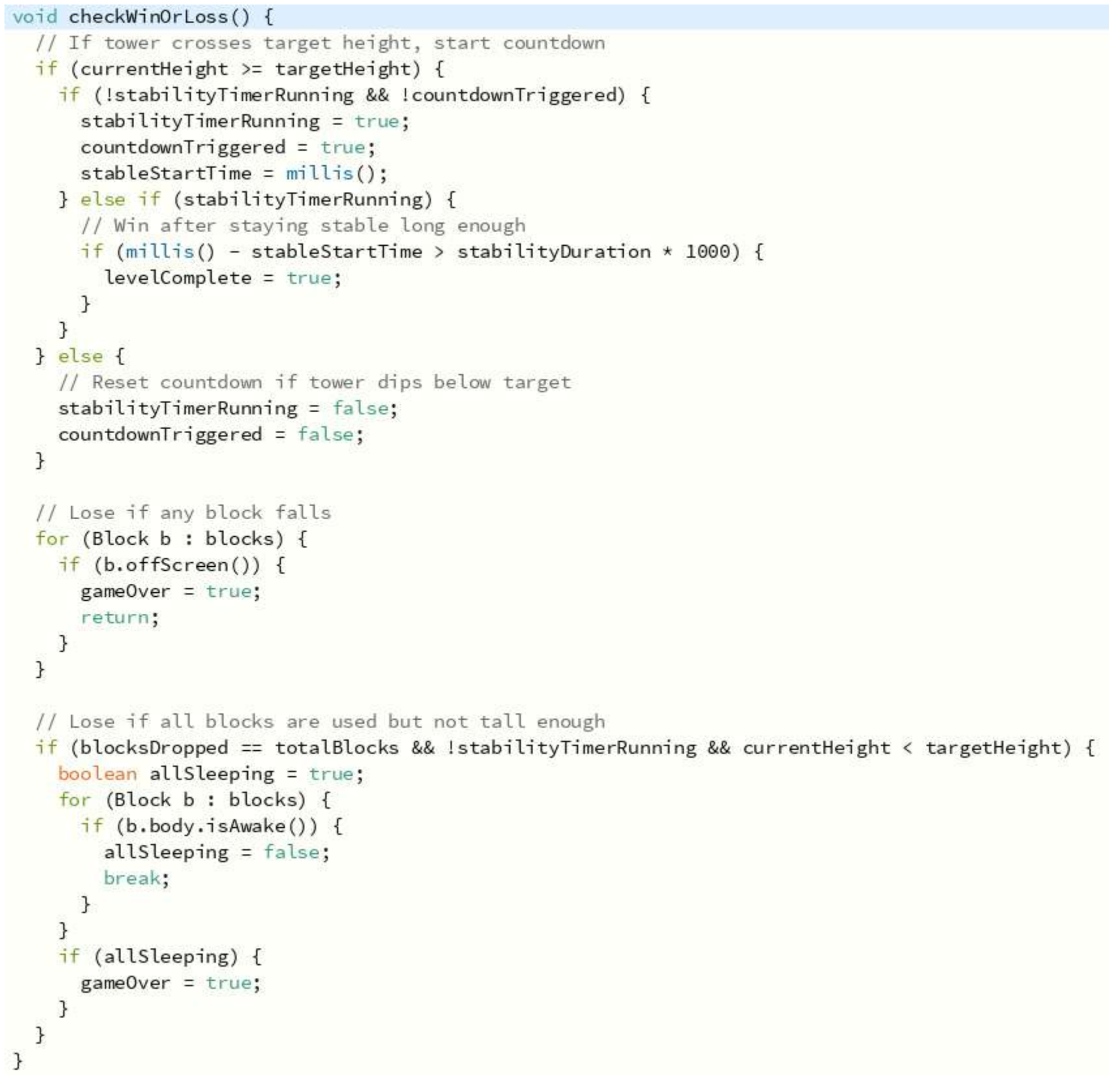


Tower reaches the target, and countdown begins



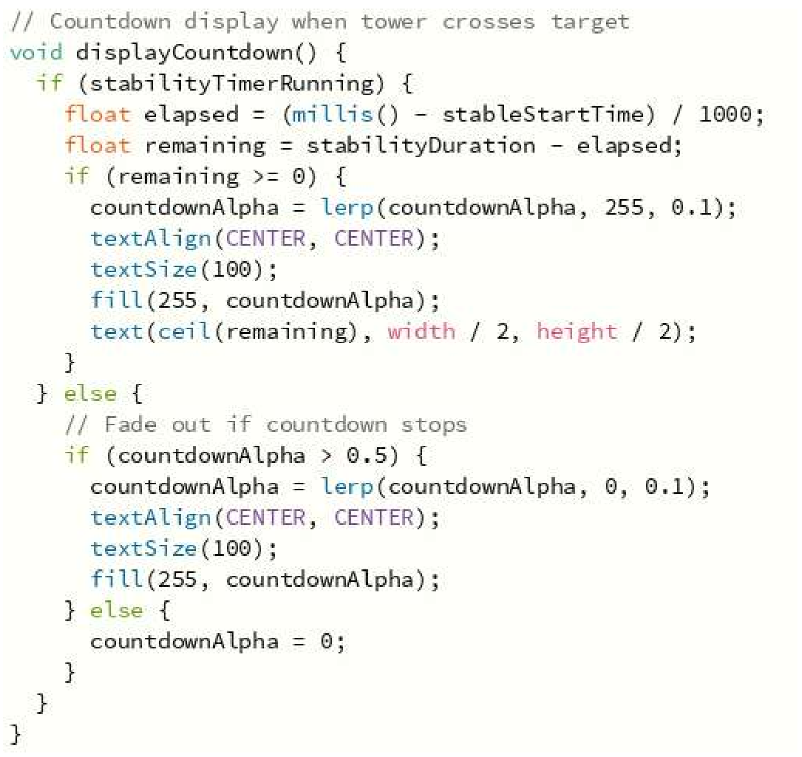
```
// Calculate the stable height of the tower
void calculateTowerHeight() {
  if (blocks.isEmpty()) {
    currentHeight = 0;
    return;
  }

  float minY = height; // Find top block position

  // Loop through all blocks
  for (Block b : blocks) {
    // Skip blocks still moving
    if (b.body.isAwake()) {
      continue;
    }

    // Find highest stable block
    Vec2 pos = box2d.getBodyPixelCoord(b.body);
    if (pos.y < minY) {
      minY = pos.y;
    }
  }

  // If no stable block, height is 0
  if (minY == height) {
     currentHeight = 0;
     return;
  }

  // Convert screen Y to tower height from ground
  currentHeight = height - minY;
}
```

Code snippet to calculate the stable height of the tower



```
void checkWinOrLoss() {
  // If tower crosses target height, start countdown
  if (currentHeight >= targetHeight) {
    if (!stabilityTimerRunning && !countdownTriggered) {
      stabilityTimerRunning = true;
      countdownTriggered = true;
      stableStartTime = millis();
    } else if (stabilityTimerRunning) {
      // Win after staying stable long enough
      if (millis() - stableStartTime > stabilityDuration * 1000) {
        levelComplete = true;
      }
    }
  } else {
    // Reset countdown if tower dips below target
    stabilityTimerRunning = false;
    countdownTriggered = false;
  }

  // Lose if any block falls
  for (Block b : blocks) {
    if (b.offScreen()) {
      gameOver = true;
      return;
    }
  }

  // Lose if all blocks are used but not tall enough
  if (blocksDropped == totalBlocks && !stabilityTimerRunning && currentHeight < targetHeight) {
    boolean allSleeping = true;
    for (Block b : blocks) {
      if (b.body.isAwake()) {
        allSleeping = false;
        break;
      }
    }
    if (allSleeping) {
      gameOver = true;
    }
  }
}
```

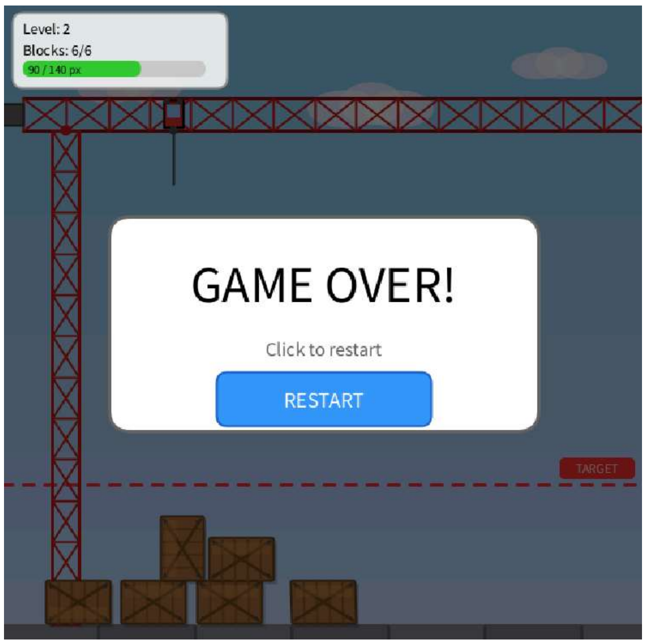
Code snippet to check if the player won or lost



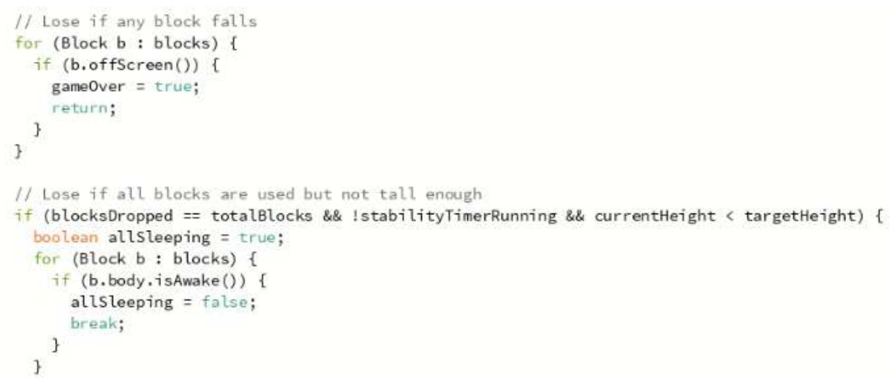
```
// Countdown display when tower crosses target
void displayCountdown() {
  if (stabilityTimerRunning) {
    float elapsed = (millis() - stableStartTime) / 1000;
    float remaining = stabilityDuration - elapsed;
    if (remaining >= 0) {
      countdownAlpha = lerp(countdownAlpha, 255, 0.1);
      textAlign(CENTER, CENTER);
      textSize(100);
      fill(255, countdownAlpha);
      text(ceil(remaining), width / 2, height / 2);
    }
  } else {
    // Fade out if countdown stops
    if (countdownAlpha > 0.5) {
      countdownAlpha = lerp(countdownAlpha, 0, 0.1);
      textAlign(CENTER, CENTER);
      textSize(100);
      fill(255, countdownAlpha);
    } else {
      countdownAlpha = 0;
    }
  }
}
```

Code snippet to display the countdown once the tower crosses the target

#4 Game over/ tower falls

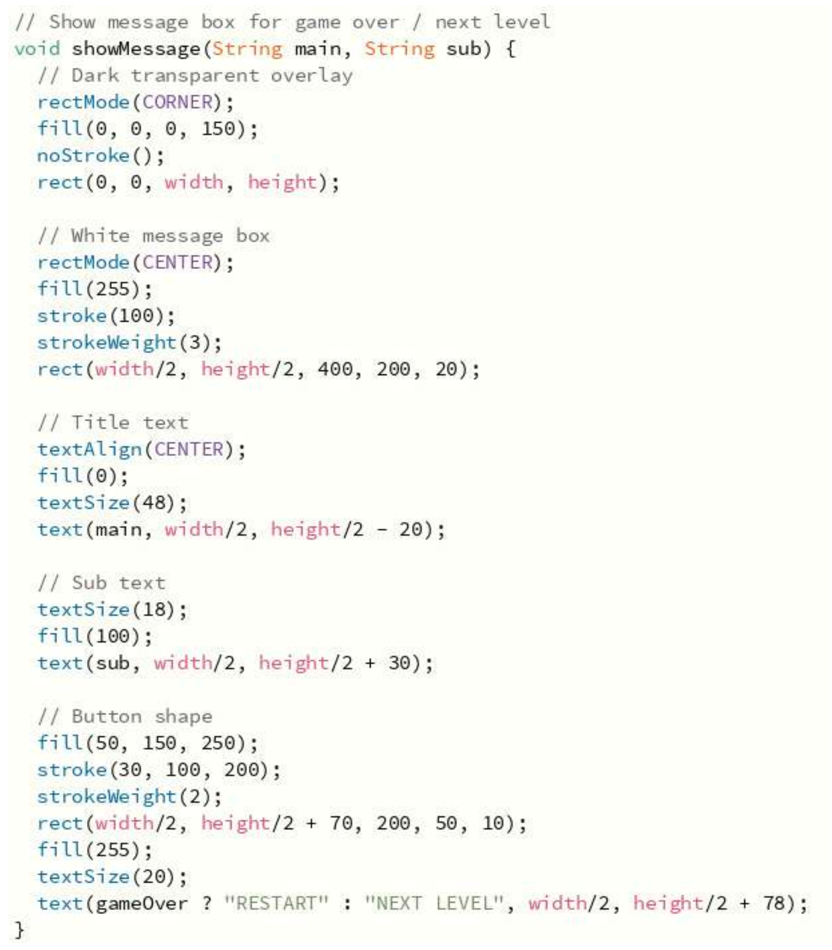


Game over if the tower does not cross the target



```
// Lose if any block falls
for (Block b : blocks) {
  if (b.offScreen()) {
    gameOver = true;
    return;
  }
}

// Lose if all blocks are used but not tall enough
if (blocksDropped == totalBlocks && !stabilityTimerRunning && currentHeight < targetHeight) {
  boolean allSleeping = true;
  for (Block b : blocks) {
    if (b.body.isAwake()) {
      allSleeping = false;
      break;
    }
  }
```

Code snippet for loss condition



```
// Show message box for game over / next level
void showMessage(String main, String sub) {
  // Dark transparent overlay
  rectMode(CORNER);
  fill(0, 0, 0, 150);
  noStroke();
  rect(0, 0, width, height);

  // White message box
  rectMode(CENTER);
  fill(255);
  stroke(100);
  strokeWeight(3);
  rect(width/2, height/2, 400, 200, 20);

  // Title text
  textAlign(CENTER);
  fill(0);
  textSize(48);
  text(main, width/2, height/2 - 20);

  // Sub text
  textSize(18);
  fill(100);
  text(sub, width/2, height/2 + 30);

  // Button shape
  fill(50, 150, 250);
  stroke(30, 100, 200);
  strokeWeight(2);
  rect(width/2, height/2 + 70, 200, 50, 10);
  fill(255);
  textSize(20);
  text(gameOver ? "RESTART" : "NEXT LEVEL", width/2, height/2 + 78);
}
```

Code snippet to display overlay if user wins or loses