

COMPREHENSIVE COMPUTER SCIENCE, ROBOTICS & GAME LAB MASTER GUIDE

LESSONS 1-20 REVISION WORKBOOK



Comprehensive Computer Science, Robotics & Game Lab Master Guide

COMPUTER SCIENCE, LITERACY & ROBOTICS (LESSONS 1-20)

MASTER STUDY GUIDE & DETAILED TECHNICAL THEORY NOTES

Grade / Level

Introductory Computer Science, Robotics & Game
Design

Duration

60 Minutes

Total Marks

50 Marks

Contents Flow	Workbook Focus
1	Test Scope & Weighting Table
2	Unit 1: Hardware & Computer Systems Theory
3	Unit 2: Game Lab Fundamentals, Grid, Coordinates, Memory & Sprites
4	Unit 3: Logic & Execution, Draw Loop, Counters, Velocity & Conditionals
5	Unit 4: Graphics, Rendering, Tinting, Transparency & Z-Order
6	Sample Practice Questions, Comprehensive Solutions & Final Checklist

Test Scope & Weighting Table (50 Marks Breakdown)

Platform: Code.org Game Lab (Interactive Animations & Games, Lessons 1–20)

Section	Topic	Marks	Weighting
Section 1: Computer Literacy & Systems Theory	Hardware Architecture & Memory Systems	5 Marks	10%
Section 1: Computer Literacy & Systems Theory	Networks, GUI & Cyber Security	5 Marks	10%
Section 2: Code.org Block Fundamentals & Sprites	Canvas Coordinate System & Origin	4 Marks	8%
Section 2: Code.org Block Fundamentals & Sprites	Sprite Declaration, Memory & Property Controls	8 Marks	16%
Section 2: Code.org Block Fundamentals & Sprites	Draw Loop Mechanics & Display Logic	8 Marks	16%
Section 3: Logic, Movement & Key Controls	Counter Patterns & Positional Math	8 Marks	16%
Section 3: Logic, Movement & Key Controls	Conditionals & Key Inputs	6 Marks	12%
Section 3: Logic, Movement & Key Controls	Velocity Properties & Collision Logic	6 Marks	12%
Total	All assessed content	50 Marks	100%

Difficulty Breakdown

- **40% Easy** — Direct recall, basic syntax
- **40% Medium** — Code prediction, property applications
- **20% Hard** — Multi-step conditional logic, edge cases

Unit 1: Computer Hardware & Literacy Theory

1. Core Hardware Architecture

Component	Definition and Function
CPU (Central Processing Unit)	The primary “brain” that executes software instructions.
RAM (Random Access Memory)	Volatile temporary working memory. Clears completely when powered off.
Storage (HDD / SSD)	Non-volatile permanent storage used to retain files, system files, and programs.
Input Devices	Keyboard, mouse, gamepad, sensors (sends environment/user data INTO the computer).
Output Devices	Monitor, projector, printer, speakers, motors (delivers processed data OUT to user).

Key rule: hardware versus software

Hardware is the physical equipment used by a computer system. Software consists of the instructions and programs that the hardware executes. The CPU executes software instructions, RAM provides temporary working space, and storage retains programs and files.

2. Digital Data Hierarchy

Level	Meaning
Bit	0 or 1
Byte	8 bits
Kilobyte (KB)	Larger unit of digital data
Megabyte (MB)	Larger unit of digital data
Gigabyte (GB)	Larger unit of digital data
Terabyte (TB)	Larger unit of digital data

Digital data hierarchy

Bit (0 or 1) < Byte (8 bits) < Kilobyte (KB) < Megabyte (MB) < Gigabyte (GB) < Terabyte (TB)

3. Networks & Security Basics

Concept	Explanation
LAN (Local Area Network)	Connects devices over a small physical area (e.g., school lab).
WAN (Wide Area Network)	Connects devices over long geographic distances (e.g., the Internet).
HTTPS	Secure encrypted web protocol (indicated by padlock icon in browser address bar).
Phishing	Social engineering attacks disguised as legitimate communication to steal personal data.

Revision reminder

RAM is temporary and volatile; HDD / SSD storage is permanent and non-volatile. LAN is local, WAN is wide-area. HTTPS protects communication with encryption, while phishing attempts to trick users into revealing personal data.

Unit 2: Detailed Game Lab Fundamentals

1. THE 2D COORDINATE SYSTEM IN GAMELAB

Computers display graphics using a 2D grid of pixels. In standard mathematics (Cartesian plane), the origin (0,0) is at the bottom-left, and Y increases as you go UP.

CRITICAL DIFFERENCE IN COMPUTER GRAPHICS

In Game Lab (and most computer graphics engines), the origin point (0,0) is located at the TOP-LEFT corner of the screen.

X-Axis (Horizontal): Measures distance from left to right.

x = 0: Far Left edge of the screen.

x = 200: Exact Horizontal Center.

x = 400: Far Right edge of the screen.

Y-Axis (Vertical): Measures distance from top to bottom.

y = 0: Far Top edge of the screen.

y = 200: Exact Vertical Center.

y = 400: Far Bottom edge of the screen.

Coordinate diagram

TOP-LEFT (0,0) → increasing X → RIGHT (400,0)

↓ increasing Y

BOTTOM-LEFT (0,400) → RIGHT (400,400)

2. UNDERSTANDING SPRITES & MEMORY ALLOCATION

What is a Sprite?

A sprite is an interactive, 2D graphic object stored in computer memory. Unlike a simple drawn shape (like a rect or ellipse), a sprite possesses properties (variables attached to it) such as position, size, rotation, visibility, and velocity.

Creating a Sprite (The 3-Step Workflow)

1. Step 1: Declare and Initialize the Sprite

```
var bot = createSprite(200, 300)
```

“var bot” allocates a space in memory named “bot”.

“createSprite(200, 300)” places the sprite’s CENTER point at X=200, Y=300.

1. Step 2: Assign an Animation/Costume Asset

```
bot.setAnimation("rover_img")
```

Links a visual asset designed in the Animation Tab (“rover_img”) to the sprite object.

1. Step 3: Modify Initial Properties

```
bot.scale = 0.5 // Scales sprite size (1.0 = 100%, 0.5 = 50% size, 2.0  
= 200% size)  
bot.rotation = 45 // Rotates sprite clockwise by 45 degrees
```

CRITICAL RULE – SPRITE DECLARATION LOCATION

Always declare sprites OUTSIDE and ABOVE the draw loop. If you put `var bot = createSprite()` INSIDE the draw loop, the computer will create 30 BRAND NEW sprites every single second, filling up computer memory (RAM) and causing severe lag or crashes.

3. SPRITE PROPERTY MATRIX

Dot notation uses the format `spriteName.propertyName`.

Property	Purpose	Typical Values
sprite.x	Horizontal position of sprite center	0 to 400 (pixels)
sprite.y	Vertical position of sprite center	0 to 400 (pixels)
sprite.scale	Multiplier for visual size	1.0 (normal), 0.5 (half), 2.0 (2x)
sprite.rotation	Clockwise rotation in degrees	0 to 360 (degrees)
sprite.visible	Shows or hides the sprite	true (visible) / false (hidden)
sprite.tint	Overlay color shade over sprite	Color strings e.g. "red", "blue"
sprite.alpha	Transparency of the sprite	0 (invisible) to 255 (opaque)
sprite.velocityX	Automated horizontal speed	Positive (right), Negative (left)
sprite.velocityY	Automated vertical speed	Positive (down), Negative (up)

Unit 3: Detailed Logic & Execution

1. THE DRAW LOOP MECHANICS (ANIMATION ENGINE)

Animation is an optical illusion created by rapidly displaying a sequence of still images (frames).

In Game Lab, function `draw()` { ... } acts as the animation engine.

Execution Speed

The draw loop runs automatically 30 TIMES PER SECOND (30 Frames Per Second / FPS).

Draw-loop execution order

1. Clear the Canvas → 2. Update Sprite Properties & Mathematical Logic → 3. Draw All Active Sprites

The Execution Sequence Inside Draw Loop:

1. **Clear the Canvas (Top of Loop):** `background("white")` wipes away the previous frame's drawings. Without this, moving objects leave a permanent trail.
2. **Update Sprite Properties & Mathematical Logic (Middle of Loop):** `bot.x = bot.x + 2` modifies sprite coordinates or checks for user input.
3. **Draw All Active Sprites (Bottom of Loop):** `drawSprites()` renders all sprite objects onto the canvas at their updated coordinates.

2. POSITIONAL COUNTER PATTERNS (MANUAL MOVEMENT)

To make an object move across the screen, its position must change slightly in every frame. This is called a "Counter Pattern": $\text{Variable} = \text{Variable} + \text{Amount}$.

Move Right: `bot.x = bot.x + 3` (Increases X coordinate)

Move Left: `bot.x = bot.x - 3` (Decreases X coordinate)

Move Down: `bot.y = bot.y + 3` (Increases Y coordinate – down towards 400)

Move Up: `bot.y = bot.y - 3` (Decreases Y coordinate – up towards 0)

Diagonal Movement:

```
bot.x = bot.x + 3
```

```
bot.y = bot.y + 3 // Moves Down-Right diagonally
```

3. AUTOMATED VELOCITY PROPERTIES

Instead of manually writing counter patterns in every frame, Game Lab provides built-in velocity properties.

- `bot.velocityX = 4`: Automatically adds 4 to `bot.x` every frame.
- `bot.velocityX = -4`: Automatically adds -4 to `bot.x` every frame.
- `bot.velocityY = 4`: Automatically adds 4 to `bot.y` every frame (moves down).
- `bot.velocityY = -4`: Automatically adds -4 to `bot.y` every frame (moves up).

Acceleration (Speeding Up):

If you increase velocity inside the draw loop, the object accelerates:

```
function draw() {  
  background("white")  
  bot.velocityX = bot.velocityX + 0.1 // Speeds up over time  
(horizontal thrust)  
  drawSprites()  
}
```

4. CONDITIONALS & BOOLEAN LOGIC (DECISION MAKING)

Programs become interactive when they can make decisions based on conditions. An “If Statement” evaluates a condition as either TRUE or FALSE (Boolean logic).

Syntax Structure:

```
if ( condition ) {  
  // Code executes ONLY when condition is TRUE  
}
```

Keyboard Input Functions:

- `keyDown("space")`: Returns TRUE continuously as long as the spacebar is held down.
- `keyWentDown("space")`: Returns TRUE for ONLY ONE FRAME when the key is first pressed.
- `keyWentUp("space")`: Returns TRUE for ONLY ONE FRAME when the key is released.

Player Movement Control Code:

```
if ( keyDown("right") ) {  
    bot.x = bot.x + 4  
}  
if ( keyDown("left") ) {  
    bot.x = bot.x - 4  
}  
if ( keyDown("up") ) {  
    bot.y = bot.y - 4  
}  
if ( keyDown("down") ) {  
    bot.y = bot.y + 4  
}
```

5. BOUNDARY DETECTION & SCREEN WRAPPING

Preventing sprites from leaving the screen requires conditional boundary checks.

Pattern A: Screen Wrap-Around (Teleportation)

If a sprite moves off the right side of the canvas ($x > 400$), reset it to the left side ($x = 0$):

```
if ( bot.x > 400 ) {  
    bot.x = 0  
}
```

Pattern B: Bouncing / Direction Reversal

If a sprite hits the right wall, reverse its speed direction:

```
if ( bot.x > 380 ) {  
    bot.velocityX = -4 // Reverses direction to move left  
}
```

Unit 4: Graphics, Rendering & Z-Order (Layering)

1. RENDERING ORDER (THE PAINTER'S ALGORITHM)

Computers draw shapes and sprites in the exact order they appear in the code (from top to bottom). This is known as the Painter's Algorithm.

Layering rule

Whatever is drawn LAST will appear ON TOP of whatever was drawn FIRST.

Sprite layering diagram

FIRST: background or shape layer → NEXT: sprite layer → LAST: later-drawn objects appear on top.

Code Example 1 (Incorrect background placement):

```
drawSprites()  
background("white")
```

Result: The background covers up all sprites, making the screen completely blank.

Code Example 2 (Layering Objects):

```
fill("yellow")  
rect(100, 100, 200, 200) // Drawn first (Background layer)  
drawSprites() // Drawn second (Foreground layer)
```

Result: Sprites appear properly on top of the yellow rectangle.

2. SHAPE & GRAPHICS STYLING BLOCKS

Block	Function
<code>background("color")</code>	Fills the entire canvas with a solid color.
<code>fill("color")</code>	Sets the interior color for shapes drawn AFTER this block.
<code>noFill()</code>	Makes shapes hollow (no inner color).
<code>stroke("color")</code>	Sets the border color for shapes.
<code>strokeWidth(number)</code>	Sets the thickness of shape borders in pixels.
<code>rect(x, y, w, h)</code>	Draws a rectangle at (x,y) with width (w) and height (h).
<code>ellipse(x, y, w, h)</code>	Draws an oval/circle centered at (x,y).

Tinting

Use `sprite.tint` to apply an overlay color shade over a sprite. Color strings include “red” and “blue”.

Transparency

Use `sprite.alpha` to control transparency: 0 is invisible and 255 is opaque.

Rendering revision prompt

- ☐ Predict which object appears on top when two drawing commands are placed in different orders.
- ☐ Explain why `background("white")` belongs at the top of the draw loop.
- ☐ Use the styling blocks to describe the fill, border, shape, tint, and transparency of a game object.

Sample Practice Questions & Comprehensive Solutions

SAMPLE QUESTION 1 (Computer Literacy - Memory)

Question

Explain why Random Access Memory (RAM) is classified as “volatile” memory, while a Solid State Drive (SSD) is “non-volatile”. [2 Marks]

Answer:

RAM is volatile because it requires continuous electrical power to store data; as soon as the computer is powered off, all data in RAM is erased. An SSD is non-volatile because it permanently retains saved files, programs, and system data even when power is disconnected.

SAMPLE QUESTION 2 (Game Lab Coordinates)

Question

A student wants to place a target sprite in the exact center of the top half of a standard Game Lab screen. What (X, Y) coordinates should be used? [2 Marks]

Answer:

X = 200, Y = 100

Explanation: The horizontal center of the 400-wide canvas is X=200. The top half spans Y=0 to Y=200, so the center of the top half is Y=100.

Notes & working space

1. _____
2. _____
3. _____

Practice strategy

1. **Identify the topic:** memory, coordinates, sprites, logic, or rendering.
2. **Quote the evidence:** use the correct definition, coordinate rule, property, or code behavior.
3. **Show the reasoning:** explain each change in position, condition, or execution order.

SAMPLE QUESTION 3 (Code Analysis — Memory Leak)

Question

Identify the flaw in the following code block and explain how to fix it:

```
function draw() {  
  background("black")  
  var coin = createSprite(200,0)  
  coin.velocityY = 3  
  drawSprites()  
}
```

Answer & Fix:

Flaw: The sprite creation block `var coin = createSprite(200,0)` is placed INSIDE the draw loop. Because the draw loop executes 30 times per second, this creates 30 new sprite objects every second, causing severe memory lag and preventing the coin from moving properly.

Correction: Move `var coin = createSprite(200,0)` and `coin.velocityY = 3` OUTSIDE and ABOVE the draw loop so the sprite is created only once.

SAMPLE QUESTION 4 (Logic & Movement Prediction)

Question

Examine this program snippet:

```
var rover = createSprite(200, 390)

function draw() {
  background("white")
  if ( keyDown("up") ) {
    rover.y = rover.y - 5
  }
  if ( rover.y < 0 ) {
    rover.y = 400
  }
  drawSprites()
}
```

Describe the exact behavior of the rover sprite when the player holds down the UP arrow key continuously for 5 seconds. [3 Marks]

Answer:

1. **When the UP arrow is held down**, `rover.y = rover.y - 5` decreases the Y coordinate frame-by-frame, causing the rover to move straight upward towards the top edge of the screen at a speed of 5 pixels per frame.
2. **When the rover passes the top edge of the canvas (`rover.y < 0`)**, the second conditional triggers.
3. **The condition `rover.y = 400`** instantly teleports the rover back to the bottom edge of the screen (screen wrap), where it continues moving upward smoothly as long as the key remains held.

Final Learner Revision Checklist

Before the test, confirm that you can:

- ☐ Distinguish hardware versus software.
- ☐ Explain volatile versus non-volatile memory.
- ☐ Identify input versus output devices.
- ☐ Compare LAN versus WAN.
- ☐ Explain HTTPS and phishing.
- ☐ Use the Game Lab top-left origin.
- ☐ Explain x/y coordinate direction: X increases left to right and Y increases top to bottom.
- ☐ Declare a sprite outside and above draw.
- ☐ Use sprite properties including x, y, scale, rotation, visible, tint, alpha, velocityX, and velocityY.
- ☐ Explain draw-loop order: clear the canvas, update logic, then draw sprites.
- ☐ Write counter patterns for horizontal, vertical, and diagonal movement.
- ☐ Use velocity to automate movement and explain acceleration.
- ☐ Differentiate keyDown, keyWentDown, and keyWentUp.
- ☐ Use boundary wrap and bouncing / direction reversal.
- ☐ Apply the Painter's Algorithm and identify Z-order from code order.
- ☐ Use the sample question strategies: identify the topic, quote the evidence, and show the reasoning.

Final notes and practice

1. _____
2. _____
3. _____
4. _____
5. _____