technology

Finish, test and share

Lesson at a glance

Pairs agree two design goals for their own Galaxy Ghosts version, then build it step by step in a new MakeCode Arcade project: blue ship at the bottom, falling ghost enemies driven by a speed variable, A-button projectiles, overlap scoring, a health status bar and game over. After each piece they run and debug on device. Wrap-up tests the finished game against those two goals; a hands-off Code talk names one bug fixed and one change they would make next time.

Learning objectives

- Finish the game and test it against the original Design Brief plan
- Share the game and explain one design or code change made during the build
- Reflect on the process of creating and debugging the program, not only the product

Before the bell — prep

Book one device per pair with browser access to arcade.makecode.com. Scrap paper for each pair to jot two design goals. Confirm the Status Bar extension loads on the school network. Open a blank Arcade project on the IWB ready to model.

Materials

Item	Qty	Per	Source	Low-cost substitute
tablet or laptop with MakeCode Arcade	1	pair	school devices	run the lesson on the IWB with pupils taking turns

Lesson 25

Teaching moves

- **What we're building:** Frame this as a finishing sitting: the aim is a working game tested against their plan, not endless new features. Model the first new piece on the IWB, then circulate so both partners take turns on the keyboard.
- **Introduction to Galaxy Ghosts:** Read the overview aloud so pairs picture the whole target: steerable ship, falling ghosts, shooting, score and health. Have each pair agree two things their version should do by the end and jot them on scrap paper — they will re-check these at Wrap Up.
- **Start a New Project:** Walk the room and check every screen shows an empty Arcade project before any blocks go down. Nobody carries work forward from another build.
- **Create your player:** Model choosing the blue space ship on the IWB. Ask 'which sprite is our player?' and stop any pair who skips the image and ends up with a blank sprite.
- **Move left and right:** Demonstrate testing joystick or arrow keys straight away. If the ship is off-screen or stuck, run it on the board and confirm left/right before anyone moves on.
- **Create the targets:** Model picking the ghost from the gallery. Ask how often a new ghost appears (every 1 second). Watch for pairs who forget to assign an enemy image.
- **Random position at the top:** Run the board build and look for ghosts appearing across the top edge. If they stack in one spot, the random-position code is missing — re-add it together before pairs continue.
- **Create a speed variable:** Model creating the speed variable carefully — spelling and selecting the right name matter. Ask what the variable stores. Watch for enemies that never fall because vy is not set from speed.
- **Fire!:** Model choosing a projectile sprite and pressing A in the simulator. Ask what should happen. Watch for a projectile that appears but does not travel upward, or no sprite chosen.
- **Hit the target:** This is the common bug spot. Model the overlap block on the IWB: kind Projectile and kind Enemy, and otherSprite (not mySprite) in the destroy block. Run it and confirm the score rises on a hit.
- **Add the 'status bar' extension:** Model adding the Status Bar extension first — the health blocks only appear after it loads. Stop anyone trying to add health blocks before the extension is in.
- **Take damage:** Model the enemy–player overlap: kind Enemy and kind Player, and sprite (not otherSprite) in the destroy block. Have pairs test by letting one ghost crash in and checking health drops by 10.
- **Game over:** On the board: health starts at 100, each crash takes 10, so ten hits reach 0 and the on-zero block fires. Have each pair test by taking ten hits. A bar that never hits zero usually means damage is not subtracting 10, or on-zero is set to the wrong kind.
- **Wrap Up:** Ask each pair to read their two start-of-sitting goals, run the game, and say whether both are met. Where the game differs, have them name what it does instead and what they think caused the gap. Early finishers change one feature (starting speed or crash damage) and re-test.
- **Code talk — what worked and what we'd change:** Hands off keyboards. Invite one pair who hit a real bug to tell expected vs actual vs the fix; restate their steps in clear order for the class.

Finish with three or four pairs naming one change they would make if they built Galaxy Ghosts again, and why.

What it should show

A working game: ship steers left/right at the bottom; ghosts spawn at random top positions and fall faster via the speed variable; A fires an upward projectile; projectile–ghost overlap destroys the ghost and raises score; ghost–player overlap drops health by 10 and destroys the ghost; health 0 shows Game Over. Odd results usually mean wrong sprite kind in an overlap, mySprite used instead of otherSprite, or the Status Bar extension never added.

Misconceptions & interventions

- **'I destroyed mySprite so the ship vanishes when I hit a ghost.'** — On the IWB, open the projectile–enemy overlap and point to otherSprite in the destroy block — that is the ghost that was hit, not the ship. Have them swap and re-run until only the ghost disappears and the score rises.
- **'The health blocks aren't in the toolbox — the game must be broken.'** — Show adding the Status Bar extension first; the health blocks only appear after it loads. Then rebuild the bar together and test one crash.
- **'Ghosts only fall in one spot so the random code must be wrong.'** — Run their onUpdateInterval on the board and check the set-position uses a random x across the top. If x is fixed, re-add the random range and watch ghosts spread.

Differentiation

Emerging	Developing	Proficient
• Pair at the teacher table for the overlap and Status Bar steps; teacher places the kind selectors while pupils choose the sprite images and press run. • Allow 'movement works + one ghost falls' as the checkpoint before adding fire and health.	• After a working hit, change the starting speed value once and predict whether ghosts will be harder or easier before running. • Retest game over by counting the ten hits aloud so the link from −10 to zero is clear.	• Change one feature from their design goals (starting speed or crash damage), re-run, and say whether the change matched the prediction. • In Code talk, lead with a full expected → actual → fix sequence so the class hears debugging named in order.

Cross-curricular hook

Tie the speed variable and score counter to the Maths algebra and data strands — a named value that changes as the game runs.