Hopscotch Coding - Week 3 - Endless Arcade Hopper

Overview

In this third week of the Hopscotch course the pupils will learn how to create a simple Platform game, in the vein of classic titles such as Frogger and Crossy Road. The keywords for this week will be **Arcade Hopper, Loop & Playtesting**. In this week's lesson the class will have the opportunity to be far more independent in their game design, deciding on level structure and challenges. There will however be some mandatory elements including the use of a button controller system.

Learning Objectives

- To understand and be able to use the terms **Arcade Hopper, Loop & Playtesting.**
- To begin to develop both game design and coding skills with the use of the Hopscotch App.
- To be able to traverse a 2D plane on both the X and Y axis using a new method of control.

Tools needed

- 15 Ipads

Starter Activity (10mins)

Before any activity commences with the iPads the tutor should post the keywords of the lesson on the board and give a brief explanation of their use in today's lesson e.g. **Loop** - "When something continuously repeats". Be sure to remind all of the pupils that they should use these words when discussing their work with either you or their peers. Do not forget to remind the children of the previous weeks' keywords and that they should still aim to use them. Introduce today's brief and that they are trying to emulate the game '**Crossy Road**'. If you wish, supplementary video material could be included as a prompt for the class!

Activity 1 - Coding the Button Controls (10 mins)

In their third week of coding the class should now be very familiar with the use of the variables **Change X by** and **Change Y by** when moving their character. This week will provide opportunity to use a different method of control. This will combine their knowledge of conditionals and collisions to create a classic set of character controls. Using the **….. is tapped** applied to 4 appropriate arrow sprites (there are very suitable arrow emojis) encourage the children to independently code each direction of movement. This will form the basis of their game.

Activity 2 - Adding moving sprite obstacles (20 mins)

Now that they have a character that moves across the screen the game requires both a goal and obstacles. As with all arcade hoppers the idea is to reach the other side of the screen without being hit by the moving obstacles. Therefore each game requires a set of moving obstacles. As Hopscotch does not allow sprites to move through the edges of the screen, each character will need to continuously move left and right across the screen. All of their movement should be coded under the conditional **Game Starts** and should use **Repeat Forever**, **Set Speed** (1000 is max) and **Change X By.** Encourage the class to vary each speed of the sprite to vary the difficulty of the game. Each pupil is aiming for roughly 5 sprites moving as obstacles across the screen.

Activity 3 - Coding the Character's Collision (15 mins)

Using almost identical code to our sprite in last week's lesson **(See Week 2)** the class should now code their character's return to the beginning of the hopper when it hits one of the moving obstacles. In order to best aid their learning, the class should attempt to do this work independently as they completed a very similar task in the previous week. Remind them that they will need to repeat the process for every single moving obstacle that appears on the screen. Encourage the class however to experiment with some other variables such as **Shrink by**, **Spin** or **Flip**.

Activity 4 - Including a Goal Sprite and Play-testing (15 mins)

The final activity of this lesson will be to finish the game by adding a **Goal Sprite** which triggers an indication that the player has won. This could be for instance a specific noise that triggers; again let the pupils lead in their decisions. Once this is complete encourage the class to engage in play-testing. It is important that the difference between **Debugging** (checking for errors) and **Play-testing** (testing how well the game works/is it playable/is the difficulty appropriate) is made explicit. Explain that if play-testing reveals anything about the game that is not as good as it could be, then this is the time to change it. This type of testing will become more common in the coming weeks as the class begin to work independently on their own games.

National Curriculum:

Coding: Write a simple algorithm whilst commanding sprites.

Coding: Debugging and checking accuracy of game.

Coding: Create a short game/animation using a simple visual programming language.

Coding: Beginning understand basic computing and mathematical concepts and vocabulary.

General: Numeracy, Reasoning Skills, Creative Design