Years 9/10 Level Up: Game Design

Curriculum links

Strand	Content Descriptions
Processes and Production Skills	Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases (ACTDIP040) Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics (ACTDIP039) Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language (ACTDIP041)

Learning focus

This unit of work is a modification of *Game On!*, a program for teaching algorithm and program design to years 7–8. This unit of work is intended to teach years 9–10 students basic programming, using a language more appropriate for these year levels.

Learning hook

Introduce students to the Unity game engine. Many modern games are made with Unity. It is an industry standard software that provides a platform to construct games. It allows for different programming languages to be used within the game engine, and you can adapt this unit of work into a language that you are comfortable with.

The Made with Unity website features many games developed using Unity; perhaps the most popular is Temple Run. Students can watch the Temple Run preview.

Ask students to play Temple Run for ten minutes to see who in the class can get the best score.

Open a discussion with students about the three components of games:

- media (graphics and sounds)
- physics
- commands.

Put students in groups, and ask them to rank the components by importance, with justifications for their rankings. They may use examples from other games that they have played to justify their responses.

Ask each group to convince another group of their responses. As a class, come up with a ranking.

Learning map and outcomes

Learning intentions

- Think like a programmer.
- Understand modular programs, data structures and object-oriented languages.

Success criteria

- Design an algorithm to construct a game.
- Program and construct the game.

Learning input

Learn the software

The following learning input relies heavily on the 'Roll-a-ball' tutorial in the Unity beginners program, available at the Unity website.

First, download Unity. Mac and PC versions and installation instructions are available. The free version of Unity is appropriate for student use.

Discuss with students the advantages of modularisation of programming. Programmers work by getting one element of their program right before moving onto other elements. This process is also important when learning to program: students need to learn one thing before another. Students will practise this by programming an element and then deconstructing it in order to understand it better.

Discuss with students the importance of tracking and recording errors, so that when predictable errors come up they can look to a troubleshooting guide to fix them easily. It is important that students understand that programming is a problem-solving process; not everything will work straight away. Programmers need a growth mindset.

Watch these clips about the concept of a growth mindset:

- Keeping moving forward!
- Grown mindset vs fixed mindset

'Roll-a-ball' tutorial

Ask students to sign up for a Unity account and open the Roll-a-ball tutorial. This tutorial teaches the basic components of Unity, using C#.

Watch the videos 'Setting up the game' and 'Moving the player'. Students can view the videos and code at the same time. They may only watch three or four seconds at a time, and then stop and code.

Ask students to annotate the Explaining Unity code worksheet while they are programming. Without stopping to reflect on the language, students will find it difficult to create their own game. After students watch the first two videos, use the worksheet to discuss segments of the code with them. Answers are included on the second page of the document.

Next, deconstruct the program with students in groups. You will need to break apart what each element of code does. Explain the following two important concepts:

- object-oriented programming
- variables.

Students can undertake the Classes and objects worksheet, about object-oriented programming, and the Variables worksheet, about variables.

Students complete Section 3 of the tutorial, which demonstrates how to do scoring (ie variables) and hit tests on game elements.

Deconstruct an example game such as Mario Bros. Ask students which elements from the tutorials are used within Mario Bros.

These elements include:

- hit tests when the Mario hits the ground (to stop him falling through)
- hit tests when the Mario hits a monster (he dies)
- hit tests when the Mario hits a brick (an element falls out of this)
- move left-right and up to jump (note that instructions are needed to ensure the user knows what they are doing).

Design the game

Next, students watch the film How the inventor of Mario designs a game, about the design process that Nintendo used when designing some of their games.

Students then brainstorm different things that they like about games that they play. In groups, ask them to brainstorm different things that they can integrate into their game, and start to develop control ideas and navigation of their program.

Learning construction

Interaction design

Students design a storyboard (with rough drawings) for their game. This should be annotated with notes such as what the character will do when certain events happen. The storyboard will then lead to the design of the students' algorithm.

Next, students write the properties of each object on sticky notes. They put the sticky notes onto a large piece of paper, and draw lines between objects that are related or that affect each other. On the lines, they write how each element affects another.

Algorithm design

Discuss with students the two different types of algorithms, pseudocode and flowchart. As more senior students, they should be encouraged to use pseudocode as it is easier to jump from pseudocode to code. For students who are struggling to understand pseudocode, a flowchart is a very good visual alternative.

Ask students to convert between flowchart and pseudocode on the Unity programming algorithm guide worksheet.

Model for the class the construction of an algorithm for Tetris. First, ask students to deconstruct how they play Tetris. You could ask questions such as: What keys do you press? What happens when you press each key? What are the conditions for winning? How would you lose? How do you get a score? These questions can be put up on the board as a list of rules/conditions, then an algorithm can be constructed out of this.

You could ask students: What's the first thing you would do as a programmer? (Test for a line of bricks being full.) Draw this information from students and together as a class, co-construct the algorithm on the board. View a sample Tetris algorithm. Students then convert a flowchart of Tetris to a pseudocode version.

BOSTES Method of Algorithm Descriptions is an excellent additional resource for teaching pseudocode and flowcharts.

Next, ask students to draft the algorithms that they need to complete the game.

Character design/background design
Students draft design ideas for characters and backgrounds. These are more descriptive drawings than the screen designs, and they will be brought into Unity for development of the project.

Students should team up with another group and get some feedback on their ideas. Students can use the critical friends process to guide their feedback.

Next, students model their characters and backgrounds in Unity.

Learning demo

Students exhibit their games. For long projects, students should be asked to present at different stages their progress, and to demonstrate to other students the exciting things they are doing in their game.

Learning reflection

The students' exhibition on game design could include a range of existing games, as well as the students' own game designs. Break up the year group into two groups, and ask students to vote on a 'people's choice' game design. The criteria are playability (fun) and programming (how complex the program is). Students can be awarded prizes for their games. A teacher selection can also be made, of the game that is the most playable and complex. Students are to evaluate their games against others using the Game up! Evaluation sheet scaffold.

Assessment

Outcome	Assessed through	Excellent	Satisfactory	Needs improvement
Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases (ACTDIP040)	Algorithm for game Testing process documentation	The student presents a range of algorithms that show correct branching, iteration and functions, as well as correct elements of structured algorithms (eg correct symbols/keywords). Their algorithms demonstrate the extent of the game and are clearly linked to the game plan. The student demonstrates testing of algorithms prior to programming. They find solutions and document them in the troubleshooting guide.	The student presents algorithms that show correct branching, iteration and functions, as well as mostly correct elements of structured algorithms (eg correct symbols/keywords). The student demonstrates testing of algorithms prior to programming. They find solutions and document them in the troubleshooting guide.	The student presents algorithms that have some correct features.
Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics (ACTDIP039)	User interface of game Evaluation	The game is functional, accessible and usable. It has a clear, easy-to-use, well-designed user interface. Evaluation of the functionality, usability and user interface is clear and reflective.	The game is functional, accessible and usable. It has a clear, easy-to-use user interface. Evaluation of the functionality, usability and user interface is clear.	The game is functional, accessible and usable. It has a clear, easy-to-use user interface. Description of the functionality, usability and user interface is clear.

Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language (ACTDIP041)	Game	The student demonstrates a range of control structures, and objects developed in Unity. Complex programming features are used within the game.	The student demonstrates a range of control structures, and objects developed in Unity.	The student attempts to demonstrate a sequence.