|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **STRAND** | | Knowledge and understanding | | | | Processes and production skills | | | | | | | | | | | | | | | | | | |
|  |  | | Digital Systems | | Representation  of data | | Collecting, managing and analysing data | | | | *Creating Digital Solutions by:* | | | | | | | | | | | | | | |
| Investigating and defining | | Generating and designing | | | | Producing and  implementing | | Evaluating | | Collaborating and managing | | | | |
|  | **Content Description** | | Investigate the role of hardware and software in managing, controlling and securing the movement of and access to data in networked digital systems (ACTDIK034) | | Analyse simple compression of data and how content data are separated from presentation (ACTDIK035) | | Develop techniques for acquiring, storing and validating quantitative and qualitative data from a range of sources, considering privacy and security requirements (ACTDIP036) | | Analyse and visualise data to create information and address complex problems, and model processes, entities and their relationships using structured data (ACTDIP037) | | Define and decompose real-world problems precisely, taking into  account functional and non-functional requirements and including interviewing stakeholders to identify needs (ACTDIP038) | | Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics (ACTDIP039) | | Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases (ACTDIP040) | | Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language (ACTDIP041) | | Evaluate critically how student solutions and existing information systems and policies, take account of future risks and sustainability and provide opportunities for innovation and enterprise (ACTDIP042) | | Create interactive solutions for sharing ideas and information online, taking into account social contexts and legal responsibilities (ACTDIP043) | | Plan and manage projects using an iterative and collaborative approach, identifying risks and considering safety and sustainability (ACTDIP044) | | |
| **Sequence of Lessons / Unit** | **Approx. time rq’d** | **Year** | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | CD | Achievement standard # | | CD | Achievement standard # |
| **Creating a digital game** | 20 | 9 |  |  |  |  |  |  |  |  |  | 4 |  | 5 |  | 6 |  | 6 |  | 6 |  |  | |  |  |

|  |  |  |
| --- | --- | --- |
| **Years 7 and 8 Achievement Standard** | **Years 9 and 10 Achievement Standard** |  |
| By the end of Year 8   * Students distinguish between different types of networks and defined purposes. (1) * They explain how text, image and audio data can be represented, secured and presented in digital systems. (2) * Students plan and manage digital projects to create interactive information. (3) * They define and decompose problems in terms of functional requirements and constraints. (4) * Students design user experiences and algorithms incorporating branching and iterations, and test, modify and implement digital solutions. (5) * They evaluate information systems and their solutions in terms of meeting needs, innovation and sustainability. (6) * They analyse and evaluate data from a range of sources to model and create solutions. (7)   They use appropriate protocols when communicating and collaborating online. (8) | By the end of Year 10   1. Students explain the control and management of networked digital systems and the security implications of the interaction between hardware, software and users. 2. They explain simple data compression, and why content data are separated from presentation. 3. Students plan and manage digital projects using an iterative approach. 4. They define and decompose complex problems in terms of functional and non-functional requirements. 5. Students design and evaluate user experiences and algorithms. 6. They design and implement modular programs, including an object-oriented program, using algorithms and data structures involving modular functions that reflect the relationships of real-world data and data entities. 7. They take account of privacy and security requirements when selecting and validating data. 8. Students test and predict results and implement digital solutions. 9. They evaluate information systems and their solutions in terms of risk, sustainability and potential for innovation and enterprise. 10. They share and collaborate online, establishing protocols for the use, transmission and maintenance of data and projects. |  |

**Creating a digital game**

A digital game can give students the opportunity to learn and refine their object-oriented programming (OOP) skills which is a requirement at years 9–10. Students follow a problem-solving process to design, build and evaluate a digital game. They state the digital design problem and decompose it in order to develop a solution. They create an algorithm for the game and relate this to an OOP approach. As a group or in pairs they implement a solution to build a computer game using OOP principles. Finally, they evaluate the end product (the game) and the solution.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Flow of activities | | | | | |
| Short text | Design problem  State the digital design problem and decompose it in order to develop a solution. | Algorithm  Create an algorithm for the game and relate this to an OOP approach. | Implementation  Implement a solution to build a computer game using OOP principles. | Evaluating  Evaluate the end product (the game) and a solution that incorporates OOP principles. |
| Questions to guide exploration | *Who’s the game for and what’s its purpose?* | *How can I represent my design of the solution?* | *How can I code my solution?* | *Have I met the user’s needs and how well did we perform the task as a group?* |
| AC alignment | Investigating and defining (ACTDIP038) | Generating and designing (ACTDIP039) / (ACTDIP040) | Producing and implementing (ACTDIP041) | Evaluating (ACTDIP042) |
| What’s this about? | A digital game can test a user’s skill, provide them with a challenge and/or educate the user, all using game play.  The first step is to define the problem. Defining what the game is required to do enables us to identify the functional and non-functional requirements. At this point we also need to identify if there are any constraints or factors that should influence the nature of the game or how it is developed. How will the user interact – via a keyboard using arrow keys, a mouse or game console?  State the problem in terms of requirements. This will help in the process of designing an algorithm and, following that, to write the computer program.  Imagine we want a game where a superhero completes a challenge to gain points. If the superhero fails to complete the challenge in the allotted time, they lose a life. For each challenge, rewards are available for the superhero to increase their power. Their power starts to decrease as the challenge begins.  To make this game, we need to:   * represent the superhero * represent their power * represent the time remaining * display their score * have a way to input instructions to make the superhero react in the intended way * alert the player when the superhero’s life has been lost * include a way to reset the game.   To incorporate the use of OOP language students will need to be familiar with how to use classes as a way of associating variables and functions. This will help students to clarify ideas in the design phase. Note in OOP, variables are often referred to as **attributes** and functions are referred to as **methods**.  Using the superhero game as an example, class ‘Superhero’ could include:   |  |  | | --- | --- | | **Attributes (variables)** | **Methods**  **(functions)** | | Image | Move | | Location | Collect tokens | | Power level | Gain power |   Other classes might include ‘Score display’, ‘Energy discs’ (for increasing power levels), etc.  **Note**: The programming language that students use to build the game will depend on students’ experience with general purpose programming languages. Note that Python on its own can be used to produce fairly rudimentary games in terms of a user interface. Students who want to include graphics can use the PyGame library by importing it at the start of the computer program. | An algorithm is a logical step-by-step process for stating how to create a digital solution. Algorithms are generally written as a flowchart or in pseudocode. A flowchart is a common way to visually represent an algorithm. Another relevant approach particular for games and apps is to do a storyboard which often focuses on the onscreen actions.  Pseudocode is a way of describing a set of instructions that does not have to use specific syntax. Students use structured English to express these instructions; for example, using ‘while’ and ‘endwhile’ when describing a ‘while loop’.  As they design the solution, students need to refer back to any constraints identified when they defined the problem, such as social and technical constraints. The design of the user interface and consideration of these constraints is referred to as user experience. Design of the user interface draws on design principles such as contrast, space, balance and repetition. | It is a requirement that students in years 9 and 10 use an OOP. This will require a shift from a procedural programming approach to using OOP approaches.  OOP is a style of programming that focuses on using objects to design and build applications.  In OOP, objects represent a combination of data (the variables or **attributes** of an object) and the actions that can be performed on or with that data (the functions or **methods** of the object). An [example](https://brilliant.org/wiki/classes-oop/) might be the declaration of a ‘car’ which has attributes that describe its physical nature (such as the number of doors, its colour, the size of the engine) and methods for the actions it can perform (such as accelerating, braking and turning).  The attributes and methods of an object are defined by its class. Using a class, we can organise information as attributes so we can reuse these elements when making multiple instances of that data type.  A programmer could make three instances of a superhero, such as Batman, Wonder Woman and the Hulk, using the Superhero class to store similar information that is unique to each superhero. For example, they look different, have different levels of power, etc. The programmer could also associate the appropriate information with each superhero. The methods of the superhero might be to jump and collect tokens. Collecting tokens could replenish power levels.  Once students have developed their algorithm and mapped out the classes with their associated attributes and methods they can begin to implement their design by programming. | **Product evaluation**  Rubrics can be used to assess the quality of the final product. In conjunction with the rubric, a peer review can be conducted with a suitable scaffold to provide feedback on the quality of the final product from a user’s perspective.  **Student performance evaluation**  As students progress through the task ask them to document their design solution, implementation plan, and any code reviews. Several pieces of evidence should be gathered to evaluate student performance. Peer reviews can also be a valuable tool. |
| The focus of the learning (in simple terms) | To introduce the basics of OOP, you may decide to set students the task of building a game, such as the Snake Game. They can work through the stages as a class. This lesson can be used to develop students’ familiarity with the Python environment. Students also learn about classes and how to build a game. They follow the suggested problem-solving process from defining the problem to evaluating the solution.  Ask students familiar with OOP who have good programming skills to design their own games in collaborative teams.  Students brainstorm game ideas, and identify the audience and purpose. Once they have agreed on an idea for the game, students sketch out interactions and how the game might flow. Students can easily add and modify ideas by sketching on a whiteboard or equivalent.  Students programming in Python may want to add graphics. You can give them examples of existing games such as those in Game Frame to download, play and review. By seeing how the game works and tracing the events onscreen to the particular code, students can build up and enhance their programming knowledge.  This type of activity can help students with the design of their own digital game. It is expected that before students implement their programming they decompose their own design problem and generate design ideas.  Unity is an alternative to Python. Unity might be favoured by students who want to develop a user interface with more graphics. Refer to the lesson provided. | Ask students to complete a noun–verb analysis of the description of the problem that they are solving.  For example (nouns in red, verbs in yellow):  *Select your superhero to start. Our superhero jumps from box to box to reach the life elixir (reward) to increase her life by 1. Along the way she captures energy discs to boost her power. She avoids the monster that guards the top box where the life elixir is stored. A collision with the monster results in a life lost. Three lives lost and the game is over.*  The nouns are usually the classes and the verbs are often the functions. Compare this to the identified classes and related functions. Students use this information in the development of the algorithm which may be in the form of a flowchart or written in Pseudocode.  Students complete a relationship diagram to map the interactions between sprites/characters and events in the game. This can help with algorithm development or can be compared with the algorithm.  Discuss the options to create an algorithm for the game. This could be in the form of a flowchart or written in structured English (pseudocode). | In their groups students implement the algorithm using a suitable programming language. It may help to provide a time limit for each team member to input the code; for example, each person has a 5-minute block to type the code then swaps with the next person. This way all students get a chance to experience programming rather than watching one competent student do all the inputting.  Given the students ideally use OOP principles in this task it is useful to discuss the implications of how this will impact the way they implement the program. What classes did students identify and what will this look like in the programming? Discuss the syntax of the program being used; for example, in Python programming:   * the class needs to be initialised, and attributes and methods need to be defined; then, when the individual instances of the class are created, its attributes must be initialised * first parameter for methods is ‘self’. The ‘self’ parameter is used to create member variables.   Remind students of the importance of making their objects and classes as clear and intuitive​ as possible.  As they go through their programming they can run parts of the code to test on screen to make sure the code works as expected. Debugging is a central aspect of the programming. Students may need to do a web search to find an answer to why their code may not be working and fix the error. Ask students to keep a log of what they did individually and as a group to overcome any problems. | **Product evaluation**  The primary focus of product evaluation is to make sure the product does what it is intended to do.  **Student performance evaluation**  Typically, teachers are required to report on students’ performance. Digital technology work is almost always project based and a variety of pieces of evidence are needed in order to assess students’ performance.  **Student-guided rubrics** represent a meaningful assessment exercise. They engage students not in the content that they are learning, but also the motivations behind why they are learning it. Students may, as a class, list characteristics of what would make a good project. For example, when considering their digital technologies project, they might consider what would make a good presentation of a requirements and data analysis. Some of these characteristics might be: *easy to understand, complete, relevant discussion of constraints and requirements,*and*consideration of user community*.    **Think Aloud**is a learning and assessment strategy designed to assist students to articulate their thought processes, and to help foster a supportive environment for learning.  In an example of **formative assessment**, students could be asked to test each other’s game and provide feedback to their peers, including the top three best features, as well as the top three issues.  In **summative assessment**, you could introduce intentional mistakes in a previously developed block of code. Students could be given the faulty code and asked to identify all the mistakes. |
| Supporting resources and tools, and context for use | **Guide to game design**  [Game design](https://www.youtube.com/watch?v=TOQTZ6N_eVg)  Good game design is essential for a positive player experience whether it’s a board game, video game or dice game.  **Examples to remix**  [Game Frame: Example games](https://gameframeforpygame.wordpress.com/example-games/)  Game Frame has been developed to take the excellent PyGame libraries and make them more accessible and easy to use for beginner to intermediate programmers.  **Lesson ideas**  [Makey Makey with Python: Snake Game](https://csermoocs.adelaide.edu.au/library/ProjectCaseStudy-SnakeGame.pdf" \t "_blank)  This lesson idea can be used to support students to create a Snake Game using OOP. Students learn about OOP using existing classes and objects, as well as designing and creating multiple new classes. This is a good intermediate project. Using Makey Makey is optional.  [Game Design: Think like an inventor](https://www.digitaltechnologieshub.edu.au/teachers/lesson-ideas/level-up-game-design)  This unit of work is intended to teach years 9–10 students OOP programming using the Unity programming language. | [Object decomposition](https://www.youtube.com/watch?v=SZg4VUmLBt4)  Object decomposition allows you to provide valuable feedback and guidance to students before any lines of code are written.  [Algorithms](https://www.khanacademy.org/computing/computer-science/algorithms)  Select relevant tutorials to teach about algorithms, including searching and sorting.  [What's an algorithm?](https://www.youtube.com/watch?v=6hfOvs8pY1k&feature=youtu.be)  This video explains algorithms and how they relate to creating a computer program.  [Entity relationship in a medieval game](https://www.youtube.com/watch?time_continue=1&v=KWPUgmmQb0Q)  Use this video to think about the relationship of characters and events. | **Learn about OOP**  [Wiki: OOP](https://brilliant.org/wiki/object-oriented-programming/)  This wiki provides an easy to understand example of a superhero in a game to describe the class superhero and an object within that class; for example, Batman. The example uses Python programming language.  [Review of variables and functions](https://www.youtube.com/watch?v=s7OwHFWp47o)  Digital technologies run on data. This is a review on data stored in variables and functions used to process data.  [What are objects?](https://www.youtube.com/watch?v=zPdSqq4N1_E&t=91s)  This video explains what an object is and how it relates to data in computer programming. Objects allow us to think about the way we represent complex problems by breaking the problem into groups of interacting objects, using ‘classes’ to define how each one will be created.  [More on OOP](https://www.youtube.com/watch?v=BweqCmrySfc)  This video explains how specifying the requirements has a bearing on the objects that are built  [Computer programming: What is object-oriented language?](https://www.youtube.com/watch?v=SS-9y0H3Si8)  This computer programming overview introduces and explores object-oriented language.  [Intro to OOP](https://www.youtube.com/watch?v=MTfWr7o7Q4U&t=23s)  This video is an overview of OOP, including the differences between OOP and the traditional structural approach, definitions of class and objects, and an easy coding example in C++  [OOP 1: Introduction](https://www.youtube.com/watch?v=WmNt2GF095k)  [OOP 2: Classes and objects](https://www.youtube.com/watch?v=POrU7vcKB_k)  [OOP 3: Inheritance](https://www.youtube.com/watch?v=rUlDPZ50nJE)  **Lesson ideas**  [Behaving with real class: Using a text-based language](https://www.digitaltechnologieshub.edu.au/teachers/lesson-ideas/behaving-with-real-class-using-a-text-based-language)  One challenge in teaching object-oriented principles is finding a suitable programming language. Many of these languages are too complex and their environments too confusing. This lesson sequence offers a choice of one of two approaches to address this problem.  [Fibonacci served three ways](https://www.digitaltechnologieshub.edu.au/teachers/lesson-ideas/fibonacci-served-three-ways)  Students learn to code separate modules that perform discrete functions but collectively meet the needs of the solution. They select the most appropriate algorithm based on the type of problem.  [Game Design: Think like an inventor](https://www.digitaltechnologieshub.edu.au/teachers/lesson-ideas/level-up-game-design)  This unit of work is intended to teach years 9–10 students basic programming, using the Unity programming language.  [Makey Makey with Python: Snake Game](https://csermoocs.adelaide.edu.au/library/ProjectCaseStudy-SnakeGame.pdf" \t "_blank)  This project creates the Snake Game referenced earlier, using a Makey Makey controller. Students build skill in the use of digital systems, OOP design and  implementation and data representation. Students learn about OOP using existing classes and objects, and about designing and creating multiple new classes. This is a good intermediate project.  [Python: Bouncing Balls](https://goo.gl/9zDa9M)  This project creates a program where five balls bounce at different rates, to build skill in the use of OOP design and implementation, and data representation. Students will learn about OOP using existing classes and objects, and about designing and creating multiple new classes. This is a good intermediate project.  [Game Frame for PyGame](https://gameframeforpygame.wordpress.com/)  Game Frame has been developed to take the excellent PyGame libraries and make them more accessible and easy to use for beginner to intermediate programmers. | [Pair programming](https://www.youtube.com/watch?v=vgkahOzFH2Q)  [This video can be used to discuss dos and don’ts of pair or group programming.](https://www.youtube.com/watch?v=vgkahOzFH2Q)  [The rubric](https://moodle.emu.edu/mod/page/view.php?id=62952), developed by the EMU CS School, is a good example of a programming rubric, and could be extended or simplified for your students.  [Grading rubric for group project](https://www.cmu.edu/teaching/assessment/examples/courselevel-bycollege/hss/tools/jeria.pdf)  [This rubric can be modified to suit the learner’s experience and skills and the project type.](https://www.cmu.edu/teaching/assessment/examples/courselevel-bycollege/hss/tools/jeria.pdf)  [Peer code review example](https://www.youtube.com/watch?v=ocMraYgqHvg)  This video shows one way to conduct a code review. The example uses university students, but similar principles can be applied to secondary students.  [Code review checklist: To perform effective code reviews](https://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-code-reviews/)  This checklist may provide some useful points to include when conducting a code review. |
| Assessment | **Suggested approaches**   * The completed Snake Game works as intended * Table showing classes with associated variables and functions   **Achievement standard**  **Define** and **decompose** complex problems in terms of functional and non-functional requirements. | **Suggested approaches**  Tracing the flow of interactions using an algorithm created to describe the game.  **Achievement standard**  **Design** and **implement** modular programs, including an object-oriented program, using algorithms and data structures involving modular functions that reflect the relationships of real-world data and data entities. | **Suggested approaches**  Presentation of the completed game and discussion of how the game is programmed using classes.  **Achievement standard**  **Design** and **implement** modular programs, including an object-oriented program, using algorithms and data structures involving modular functions that reflect the relationships of real-world data and data entities.  **Test** and **predict** results and implement digital solutions. | **Suggested approaches**   * Consider using a checklist for students to review their programming.   For example:   * + Am I able to understand the code easily?   + Is the code written following acceptable coding principles?   + Is the same code duplicated?   + Can I test/debug the code easily to find the cause of errors?   + Where and how well have I used ‘classes’ in the program? * A log could be made of the days worked on the project with a one-line statement about the work/task covered for each entry. A rating could be made of their input. * Student negotiated and agreed rubrics.   **Achievement standard**  **Design** and **evaluate** user experiences and algorithms. |