Designing a mini-game with variables

*Please refer to the online lesson plan on the DT Hub to access all website links and additional resources.*

### Year levels: 5–6; Duration 3-4 hours

In this learning sequence, students are guided to design a mini-game for a target audience. They use a template to plan and record their ideas for a three-level game considering the main characters, theme and goal for each level. Students program their game using Scratch or a similar visual programming platform. Sample Scratch programs are provided for students to remix.



*Image source: Maze designed by Wannapik Icons by Flaticon*

# Learning map and outcomes

**Learning intention**

To use the design of a mini-game to develop programming skills, design thinking and computational thinking.

**Success criteria**

Students will be able to:

* design a three-level game that has a clear purpose and is designed for a particular audience
* use branching and iteration to control the flow of their program
* use a range of variables for keeping score, keeping track of lives or the use of a timer.

# Preparation and materials

* Teacher slides: Game design [PPT]
* Teacher slides: Game design: programming tips [PPT]
* Worksheet: Designing a mini-game [PDF]
* Worksheet: Game design template [Word]
* Scratch 3.0 desktop version or similar visual programming platform
* Microsoft Makecode arcade

# Learning hook

Pose the question: ‘What makes a good digital game?’

To help students answer the question, provide them with a link to Microsoft Makecode arcade or a similar site that provides a range of digital games. Have students select and explore a game of interest.

Share students' thoughts on what makes a good game.

Suggestions may include:

* appealing gameplay that is fun and challenging
* easy to understand how to play
* easy-to-use controls
* good graphics.

Show the connection to real people working in game design using the poster: Game design: Phoebe Watson.

# Learning input and construction

You may decide to use the Teacher slides to support this part of the lesson. The slides provide prompts and guidance on what to consider when designing a game in a sequential and logical way.

* Discuss the steps in planning a mini-game (slides 2 to 14).
* Use an example of a mini-game to explain how to record ideas following the game design template (slides 15 to 18).
* Discuss some basic programming skills needed to code the game using slides (slides 19 to 21).

Provide a template to help students think through the details of their game, including its purpose, characters, theme and the goal for each level. Students can collaborate with a partner on this task, which will also be beneficial during programming. Partner programming allows students to share ideas, troubleshoot together, and build on each other's strengths, fostering teamwork and problem-solving skills.

*Image: Game design template*

Organise students to discuss their game design with another pair to give and receive feedback.

Differentiation

* For some students, focusing on one single level of the game will be enough of a challenge. This may help avoid possible overwhelm with the more complex task of a three-level mini-game. Use the simplified template: Designing a mini-game.
* Some students may choose to describe their three-level mini-game using a different form of representation, such as a flowchart or mind map.

# Learning demo

Working in pairs, students follow their design plan to implement their game using Scratch 3.0 or a similar visual programming language. Their game should include decision-making elements, such as If/Then statements to trigger events, and incorporate variables such as tracking lives or scoring.

Provide examples of programming and discuss how to code various elements. Use the *Teacher slides: Game design: programming tips* to guide the conversation. These slides include examples of coding concepts like loops, conditionals and variables. Encourage students to share their ideas and collaborate on developing strategies for coding different aspects of their game.

To support and guide students with how to program a variable, view this video: Conditions | Lesson 6. The video includes an introduction to variables and how to apply them in a game scoring points for a basketball shot.

Differentiation

* For students with limited programming skills, provide an alternative to Scratch such as Microsoft Makecode arcade. Students can choose a game to make following the self-paced tutorial that includes video and prompts to complete an existing game.
* Remix a Scratch game, for example, Mazes and lives. The ‘See inside’ button on the project page enables students to view the code and explore how the program runs. A useful strategy is to briefly explore the game and then predict what blocks are used to program the game. They can then remix the game.
* Remixing is a relevant teaching approach to support students to develop their coding skills. Students could:
	+ remix the game using different sprites
	+ create a second level using elements of the game
	+ apply code examples to their own new game.



*Image: Project page: Scratch maze game*

# Reflection

Share the finished games as a class. Lead a discussion on the game design activity, asking prompt questions such as those below.

* **Game design:** ‘How did you ensure that your game was engaging and fun for the player? What elements of your design contributed to the overall user experience?’
* **User input and control:** ‘What types of user inputs did your game require (for example, keyboard, mouse)? How did you decide on these inputs, and how do they enable the player's control over the game?’
* **Programming blocks (If/Then blocks, variables):** ‘How did you use If/Then blocks or variables in your game? Can you explain a specific example of how these programming elements helped you manage game events or track game progress?’

### Why is this relevant?

Designing a digital game draws on students’ computational thinking and critical and creative thinking. When creating a digital game for a specific audience, students consider user needs and how the game will meet those needs. Designing a game is an engaging way to apply branching and iteration to control the program's flow, and explore the use of variables for scorekeeping, tracking lives and timing.

# Australian Curriculum alignment

Technologies – Digital Technologies Years 5-6

Implement algorithms as visual programs involving control structures, variables and input AC9TDI6P05

English – Literacy Year 6

Examine texts including media texts that represent ideas and events, and identify how they reflect the context in which they were created AC9E6LY01

# Resources

* Teacher slides: Game design [PPT]
* Teacher slides: Game design: programming tips [PPT]
* Worksheet: Designing a mini-game [PDF]
* Worksheet: Game design template [Word]
* Scratch game example: Mazes and lives
* Microsoft Makecode arcade
* Video: Conditions | Lesson 6
* Poster: Programming infographic
* Learn more about teaching programming

# Assessment

Achievement standard

By the end of Year 6, students design algorithms involving complex branching and iteration and implement them as visual programs including variables.

Use this rubric to assess student skills, processes and knowledge.

|  | **1 (limited)** | **2 (basic)** | **3 (proficient)** | **4 (advanced)** |
| --- | --- | --- | --- | --- |
| **Knowledge of visual programming blocks** | demonstrates a limited understanding of visual programming blocks | names some visual programming blocks or describes their purpose | describes the purpose of the visual programming blocks and provides relevant examples to support their understanding | explains how programming blocks are organised by type, their purpose and provides relevant examples to support their understanding. Their examples illustrate blocks to enable input, decisions, repeat steps and data stored as a variable |
| **Identifies control structures: investigating sample visual programs** | demonstrates limited ability to describe or identify blocks in given examples and interpret the sequence of steps | identifies most block types in provided examples. Interprets the sequence of steps and identifies user input, decisions (branching) or repeat steps (iteration) | consistently identifies block types correctly in various examples. Interprets the sequence of steps, predicts outcomes of blocks and identifies user input, decisions (branching) and repeat steps (iteration) | demonstrates a high level of accuracy in identifying block types of a range of complex visual programs. Interprets the sequence of steps, predicts outcomes of blocks and identifies user input, branching and iteration. Explains ways a variable serves as a 'container' to hold and manipulate different data types within a program |
| **Producing and implementing visual programs** | with guidance, creates a program in a sequence of steps to perform a basic task with limited opportunity for user input | creates a program with user input in a sequence of logical steps that includes a decision (branching) to provide the user with a choice | creates a program with user input in a sequence of logical steps that includes multiple decisions (complex branching) and iteration (repeat steps) | creates a program with multiple user input in a sequence of logical steps that includes multiple decisions (complex branching) and iteration (repeat steps). Includes the use of variables to hold and manipulate different data types within a program |