

{oo/Think @JC >

賽馬會運算思維教育

Inspiring digital creativity 啟發數碼創意

# COMPUTATIONAL THINKING EDUCATION



LEVEL

2

Created and Funded by:



The Hong Kong Jockey Club Charities Trust

Co-created by:



香港教育大學  
The Education University  
of Hong Kong



Massachusetts  
Institute of  
Technology



香港城市大學  
City University of Hong Kong

# CoolThink@JC - Computational Thinking Education Programme

Seeking to inspire digital creativity among students and nurture their proactive use of technologies for social good from a young age, CoolThink@JC is a computational thinking education initiative created and funded by The Hong Kong Jockey Club Charities Trust, co-created by The Education University of Hong Kong, Massachusetts Institute of Technology and City University of Hong Kong, and supported by the Education Bureau of Hong Kong. In collaboration with local educators and the world's leading experts, CoolThink@JC empowers teachers with high-quality teaching materials, learning platform and professional development programmes.

CoolThink@JC aims to mainstream computational thinking education in Hong Kong, to prepare students for a fast-changing digital future through a hands-on, minds-on, joyful learning experience and be the creator of technology. Since its launch in 2016, the project has developed a three-year curriculum for upper primary school pupils with 14 class hours per school year. It is on track to support 200 primary schools and more than 600 teachers, and enable over 100,000 upper primary school

students to benefit from the computational thinking education classes. The project has supported more than 368 primary schools in different ways. At the same time, through cooperation with different school-sponsoring bodies, teachers' associations and the Education Bureau, it encourages more than 70% of the public and Direct Subsidy Scheme schools in Hong Kong to jointly promote computational thinking education. An independent evaluation has found that students participated in CoolThink@JC grew twice as much in problem-solving skills when compared with non-participating students.

CoolThink@JC received three international accolades, showing that its overall design, curriculum and pedagogical approach have reached international standards, and have far-reaching impact to Hong Kong's technology education ecosystem. The three accolades include: QS Reimagine Education Awards, International Society for Technology in Education (ISTE) international certification (Seal of Alignment - Student Standard), Education Alliance Finland certification.



# Message from The Hong Kong Jockey Club Charities Trust

---

The Hong Kong Jockey Club is dedicated to acting continuously for the betterment of our society. This is made possible by the Club's unique integrated business model, through which revenue is returned to the community in the form of tax payments and charitable donations. As one of the top ten charity donors in the world, the Club's Charities Trust supports various charitable projects to address different societal needs. With "Children & Youth Development" as one of its focus areas, the Club helps young people give full play to their strengths by developing their knowledge and capabilities. We also want to cultivate positive, compassionate, caring young individuals, who are committed, like the Club, to the betterment of our community.

As digital technology and computational thinking skills are major drivers of today's social and economic development, inspiring the digital creativity of students is among our top priorities. Yet, CoolThink@JC is more than an initiative that "helps everyone code". We believe that empowering students to move beyond technology consumption to problem-solving, creation and innovation requires them to not only acquire coding concepts and skills, but also to explore new perspectives. The aim is to gear them up to address the challenges around them with the application of digital technology as well as advance future opportunities and social good.

The CoolThink@JC project seeks to sustainably develop Hong Kong's computational education ecosystem. It is expected that by the end of 2024, the programme will have nurtured nearly 100,000 local upper primary school students. The project has developed a quality-assured, internationally-recognised curriculum, backed by evidence-based independent evaluation and research. We are delighted to make this quality teaching material accessible to all students citywide. By encouraging both teachers and students to make the best use of this resource, we aim to instill the project's concept of "to play, code, think and reflect" in daily lives.

The Club would like to extend its sincere gratitude to the Hong Kong SAR Government and various sectors of our society for their support of CoolThink@JC. Our heartfelt thanks especially go to its co-creators, The Education University of Hong Kong, Massachusetts Institute of Technology and City University of Hong Kong. We are also grateful to all supporting partners, principals, teachers and parents for their commitment to nurturing our younger generation. We look forward to your continued support of this meaningful project so it can benefit as many schools and students as possible.

# Preface from Massachusetts Institute of Technology

---

More and more, our lives are being shaped by Information Technology. To understand the basic ideas driving this change has become a fundamental skill for everyone, even young children. The sample lessons in this book are an introduction to Computational Thinking for upper level primary school students.

Designed by teams at Massachusetts Institute of Technology (MIT) and The Education University of Hong Kong, the lessons are now being used and tested in 200 primary schools as part of CoolThink@JC, an initiative of The Hong Kong Jockey Club Charities Trust. The goal of CoolThink@JC is not only to teach kids to program, but more importantly to help them be creators in a world transformed by computation, and to empower them to take action to realize their ideas for using computation to improve life for their family, friends and community.

The lessons presented here introduce Computational Thinking in three aspects: concepts, practices and perspectives. Concepts are the pieces from which computer operations are constructed. They are the programming elements that provide sequencing, repetition, responding to events, and other operations. Practices are the habits of thought that make people effective in creating computer artifacts. They are principles like abstraction, testing and incremental design. Perspectives are attitudes by which people come to see themselves as effective participants in an information society, attitudes such as appreciating which kinds of issues can be addressed using computing, or feeling that you are able to express ideas through computing.

Mastering Computational Thinking includes gaining experience with programming. The CoolThink@JC material uses two programming environments: Scratch and MIT App Inventor 2. Scratch was developed at MIT and App Inventor was initially created at Google research and later continued at MIT. Both environments are designed to make it easy for beginners to create programs. Rather than typing programming code character by character, you build programs by working with graphical blocks that you move around on a display screen and piece together like pieces of a jigsaw puzzle. Scratch emphasizes creating interactive graphics and animations that run in the computer browser. App Inventor programs produce mobile applications that you can install on smartphones or tablets and share with others. These applications let people interact with the world around them and draw upon all the power the Internet has to offer.

We're living in a time of wonders, where Information Technology can bring amazing powerful resources within the grasp of even young children. The lessons in this book are mere hints of what's possible and the barest beginning of a great adventure. Please join us on this adventure.

## **Hal Abelson**

*Professor of Computer Science and Engineering*  
MIT

# Preface from The Education University of Hong Kong

---

The Education University of Hong Kong (EdUHK) dedicates to the advancement of professional development for teachers through research. We also focus on leading curriculum development in Hong Kong through new pedagogical design. The Education University of Hong Kong was invited by The Hong Kong Jockey Club Charities Trust in 2016 to launch the CoolThink@JC programme with The Massachusetts Institute of Technology. The two universities jointly develop a set of computational thinking curriculum units suitable for grade four to grade six primary students in Hong Kong. The curriculum aims to encourage schools to provide learning opportunities for upper level primary school students, and to nurture their computational thinking via the programming activities in Scratch and MIT App Inventor environment. The objectives of CoolThink lessons are to boost students' interest in programming, equip students with essential capability to code, and enable students to cultivate their problem-solving skills and digital creativity. It also helps students to succeed in the digital era by laying the foundation for their creative thinking and pioneering spirit.

CoolThink lessons and the associated learning activities are designed and developed base on the student-centered learning approach. It enables students to master Computational Thinking Concepts through direct engagement in programming activities. More importantly, through the process of programming, we introduce important Computational Thinking Practices, broaden students' Computational Thinking Perspectives needed in the digital age, and cultivate students' understanding of how developing games and applications can bring positive impacts to the society.

EdUHK emphasizes the importance of professional development for teachers in computational thinking education. Teacher training includes pedagogy for computational thinking development of students and the follow-up reflection on teaching computational thinking upon classroom practices. The professional development for teacher is a vital part of this programme. Only when teachers have a good understanding of the pedagogy for computational thinking development, and through reflection after teaching practice can they integrate the Concepts, Practices and Perspectives into their classroom teaching and make progress. The EdUHK team modifies the computational thinking pedagogy overtime along with the development of CoolThink@JC. From using the "To play - To think - To code - To reflect" pedagogy initially to promote thinking education, the team further introduces the "seven-step" programming pedagogy to guide teachers to conduct productive computational thinking lessons for inspiring digital creativity. Soon after the commencement of the second phase of CoolThink@JC, students in Hong Kong experienced a long period of online lesson due to suspension of face-to-face teaching under the pandemic. The project team reacted quickly by making a series of animations, which are developed based on CoolThink Scratch and MIT App Inventor curriculum units. This gives raise to the pedagogy of using animations for self-directed learning, as a further enhancement of pedagogical practices for computational thinking development in response to the limited teaching time.

## **Professor Siu-Cheung Kong**

*Professor, Department of Mathematics and Information Technology,  
Director, Centre for Learning, Teaching and Technology,*

The Education University of Hong Kong

# Table of Contents

<b>CoolThink@JC Framework</b>	<b>8</b>
<b>Unit 1 - Hello It's Me</b>	<b>9</b>
1. Student Guide Lesson 1	9
2. Student Guide Lesson 2	14
<b>Unit 2 - My Piano App</b>	<b>19</b>
1. Student Guide Lesson 1	19
2. Student Guide Lesson 2	22
<b>Unit 3 - Music Jukebox App</b>	<b>27</b>
1. Student Guide Lesson 1	27
2. Music Jukebox Design Worksheet	31
3. Student Guide Lesson 2	33
4. Two Stars and a Wish Worksheet	34
5. Student Guide: Challenge	35
<b>Unit 4 - Addition Game</b>	<b>40</b>
1. Student Guide Lesson 1	40
2. Student Guide Lesson 2	49
3. Student Guide: Challenge	56
<b>Unit 5-6 - Vocabulary Learning App</b>	<b>59</b>
1. Student Guide Lesson 1	59
2. Student Guide Lesson 2	63
3. Testing and Debugging Worksheet: Lesson 2	70
4. Variables Worksheet	72
5. Student Guide: Lesson 3	75
6. Testing and Debugging Worksheet: Lesson 3	79
7. Student Guide: Lesson 4	83
8. Testing and Debugging Worksheet: Lesson 4	91
9. Student Guide: Challenge	93
<b>Unit 7-8 - Find the Gold App</b>	<b>94</b>
1. Student Guide Lesson 1	94
2. Student Guide Lesson 2	99
3. Student Guide Lesson 3	112
4. Make a Pizza with a Procedure's Instruction	120
5. Student Guide: Lesson 4	128
<b>Combo Ball Game</b>	<b>136</b>
1. Student Guide	136
2. Combo Ball Game Design Worksheet	149
3. Two Stars and a Wish Worksheet	156
4. Combo Ball Game Presentation Worksheet	157
<b>App Inventor Final Project</b>	<b>158</b>
1. MIT App Inventor Final Project Student Guide	158
2. Educational Game App Design Worksheet	167
3. Peer Feedback Worksheet	175
4. Two Stars and a Wish Worksheet	177
<b>Voice Calculator</b>	<b>178</b>
1. Student Guide	178
2. Voice Calculator Worksheet 1	194
3. Voice Calculator Worksheet 2	196
4. Voice Calculator Worksheet 3	199

# CoolThink@JC Framework

Computational thinking entails developing fluency with programming **concepts** (key constructs and ideas that are central to most forms of computing). But more fundamental than specific coding details, computational thinking includes the ability to pose problems and seek solutions that use key **practices** (the activities people engage in when creating computational projects) involved in computing and programming. And beyond solving problems with computers, young people should develop **perspectives** like digital empowerment, which enables them to see how challenges in the world around them could be addressed through computing; and computational identity, which instills in them the knowledge that they can have active, positive roles as people who can use computers to enhance their lives and the lives of people around them.

## 01

### CT Concepts



#### CT Concepts

- < Sequences / > : identify a series of ordered steps for solving a programming task
- < Events / > : one thing causing another thing to happen
- < Conditionals / > : make decisions based on conditions
- < Operators / > : support for mathematical and logical expressions
- < Parallelism / > : make things happen at the same time
- < Repetition / > : run the same sequence multiple times
- < Naming and Variables / > : store information to be referenced and computed in a program
- < Data structures / > : basic ways data are stored, retrieved, and updated
- < Procedures / > : create code blocks to modularize and abstract sequences of commands

## 02

### CT Practices



#### CT Practices: Problem-solving Skills

- < Testing and debugging / > : make sure things work, otherwise find and solve problems when they arise
- < Being incremental and iterative / > : develop a little bit, then try it out, then develop more
- < Reusing and remixing / > : make something by building on existing code, projects or ideas
- < Abstracting and modularizing / > : explore connections between the whole and the parts
- < Algorithmic thinking / > : articulate a problem's solution in well-defined rules and steps

## 03

### CT Perspectives

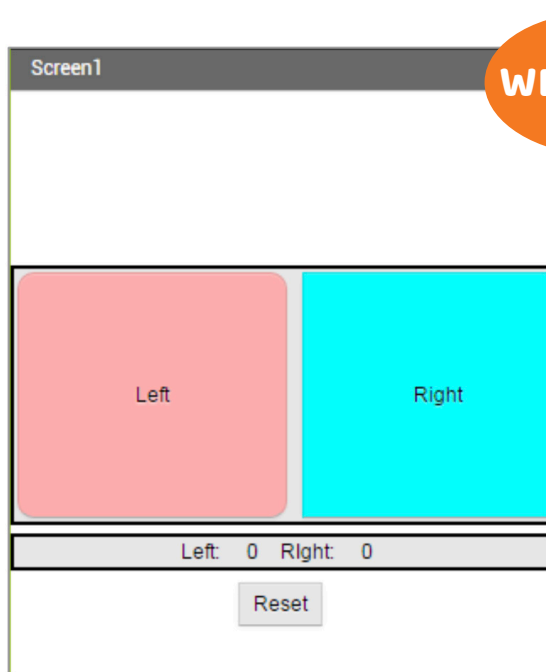


#### CT Perspectives: Identity and Motivation

- < Expressing / > : create and express ideas through this new medium
- < Questioning / > : feel empowered to ask questions about and with technology
- < Connecting / > : appreciate that others are engaged with and appreciate one's creations
- < Digital empowerment / > : develop the ability to see problems in the world as solvable through code
- < Computational identity / > : see oneself as being able to enhance the world through coding

## Lesson 1

## TWO BUTTON GAME



WELCOME!

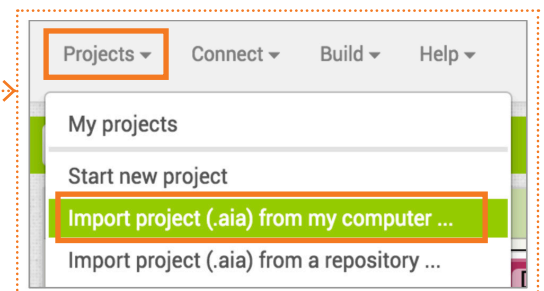
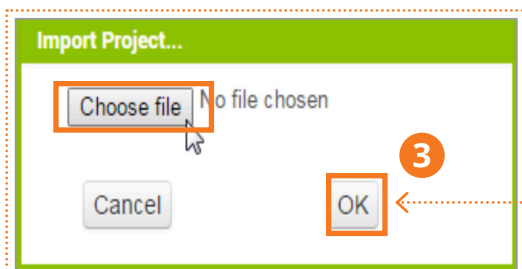


In this activity, you will use the TwoButtonGame project to learn the MIT App Inventor programming environment.



## START HERE

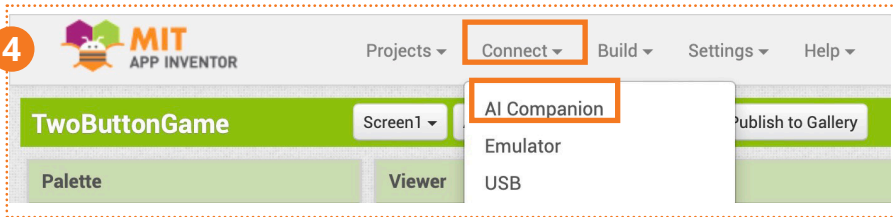
- ▶ 1 Go to the MIT App Inventor website: <http://ai2.appinventor.mit.edu>
- ▶ 2 Under **Projects**, click **Import project (.aia) from my computer.**
- ▶ 3 Choose the aia file (TwoButtonGame.aia) your teacher provides you. Click **OK**.





## LET'S RUN THE APP!

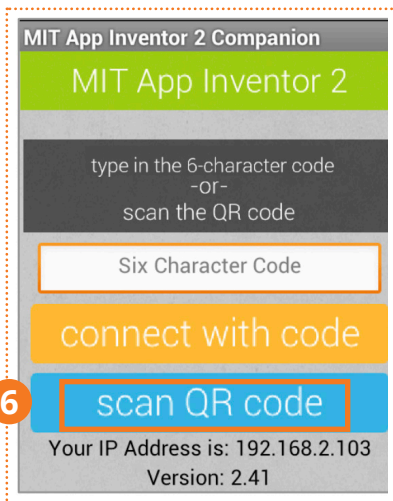
- ▶ **4** Under **Connect**, click **AI Companion**.



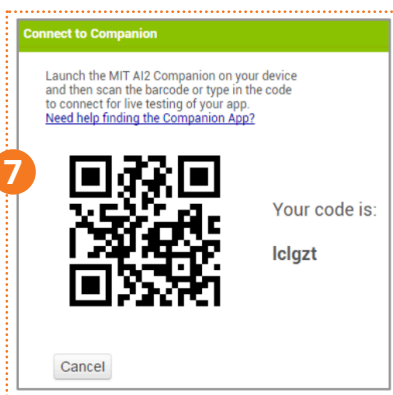
- ▶ **5** On your tablet or smartphone, run the **MIT AI2 Companion** app.



- ▶ **6** Click the **scan QR code** button.



- ▶ **7** Scan the **QR code** on the computer screen with your smartphone or tablet camera.

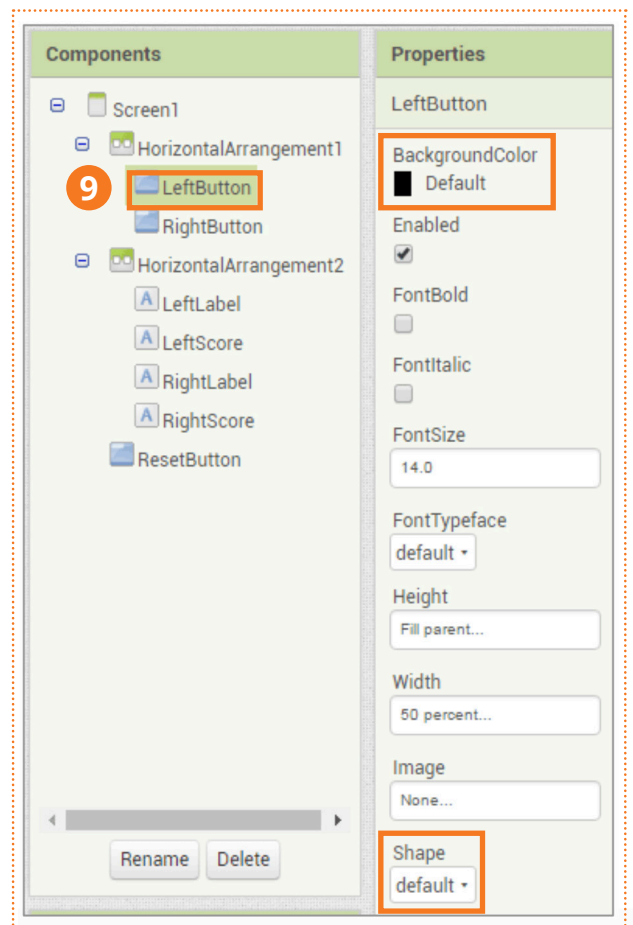


## CHANGE THE BUTTONS

- ▶ **8** Make sure you are in the Designer window by clicking on the **Designer** button in the top right corner of the window.

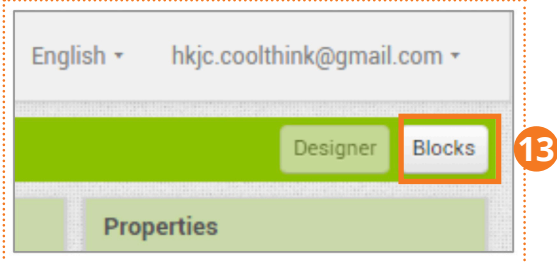


- ▶ **9** Click on **LeftButton** and change the button properties as follows:
  - ▶ **BackgroundColor**: any colour you like
  - ▶ **Shape**: rounded
- ▶ **10** Change **RightButton** in the same way.
- ▶ **11** Make other changes you want. E.g. How do you make the **ResetButton** larger?
- ▶ **12** Check the app on your smartphone or tablet. Note how the app changes when you make a change in MIT App Inventor.

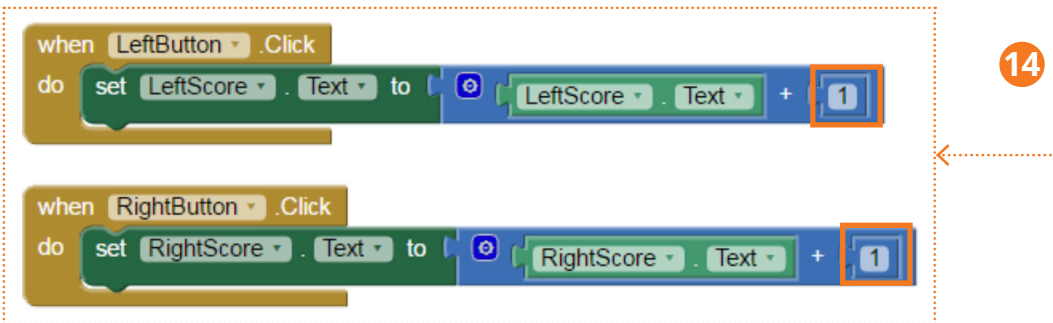


## CHANGE THE SCORE


- ▶ **13** Click the **Blocks** button and go to the Blocks Editor.



- ▶ **14** In the **when LeftButton.Click do** and **when RightButton.Click do** blocks, change the number to any number other than 1.



- ▶ **15** Play the game again on your smartphone or tablet to see the changes!



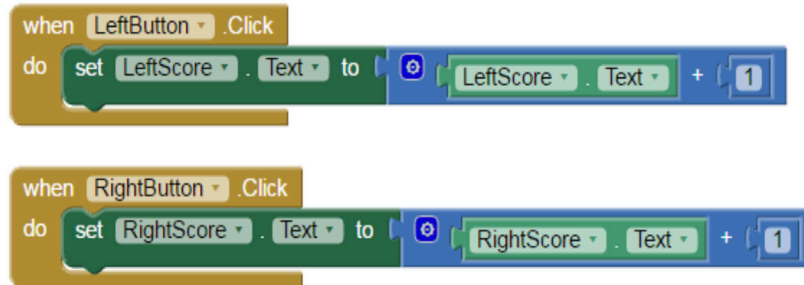
CT Tips

**when Button.Click do** is a typical **event**. An **event** is something that triggers actions.

## COMPUTATIONAL THINKING CONCEPTS

The following are the computational thinking concepts learnt in Lesson 1.

### 1 Events

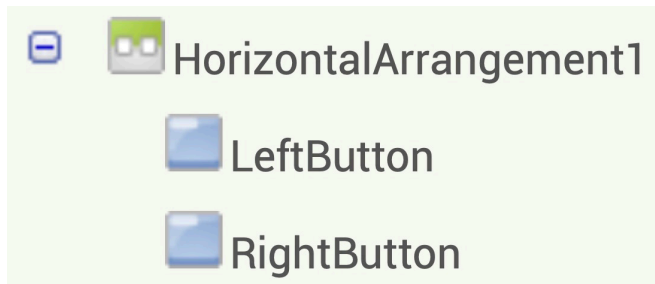


```
when LeftButton .Click
do set LeftScore . Text to LeftScore . Text + 1

when RightButton .Click
do set RightScore . Text to RightScore . Text + 1
```

The image shows two Scratch code blocks. The first block is a 'when clicked' block for 'LeftButton'. It contains a 'do' block with a 'set text to' block. The 'set text to' block has 'LeftScore . Text' in the 'to' field and 'LeftScore . Text + 1' in the 'value' field. The second block is a 'when clicked' block for 'RightButton'. It contains a 'do' block with a 'set text to' block. The 'set text to' block has 'RightScore . Text' in the 'to' field and 'RightScore . Text + 1' in the 'value' field.

### 2 Naming



```
HorizontalArrangement1
  LeftButton
  RightButton
```

The image shows a Scratch stage with three objects. The top object is a 'HorizontalArrangement1' block. Below it are two 'Button' blocks, 'LeftButton' and 'RightButton'.

## Lesson 2

## Hello it's me

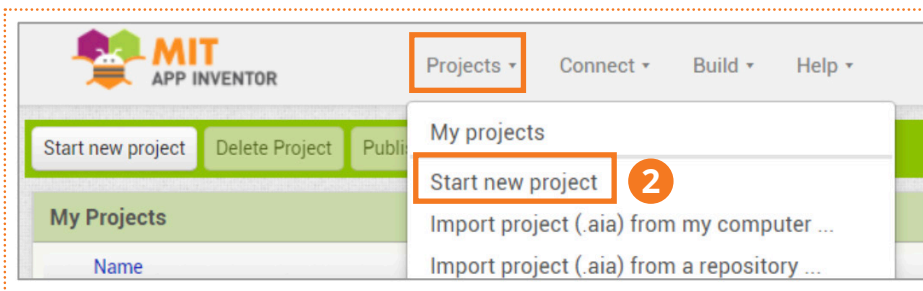


In this activity, you will create a new project with MIT App Inventor and use your own photo and voice to tell people who you are.

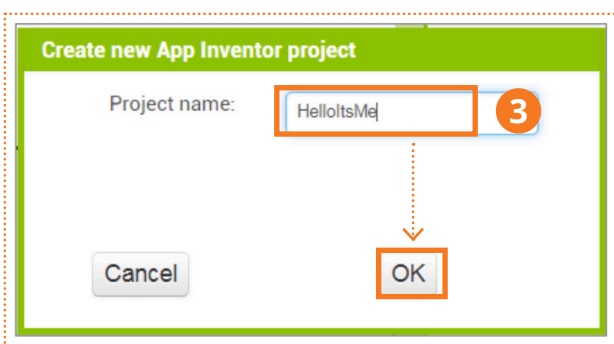


**START HERE**

- ▶ 1 Go to the MIT App Inventor website: <http://ai2.appinventor.mit.edu>
- ▶ 2 Under **Project**, click **Start new project**.

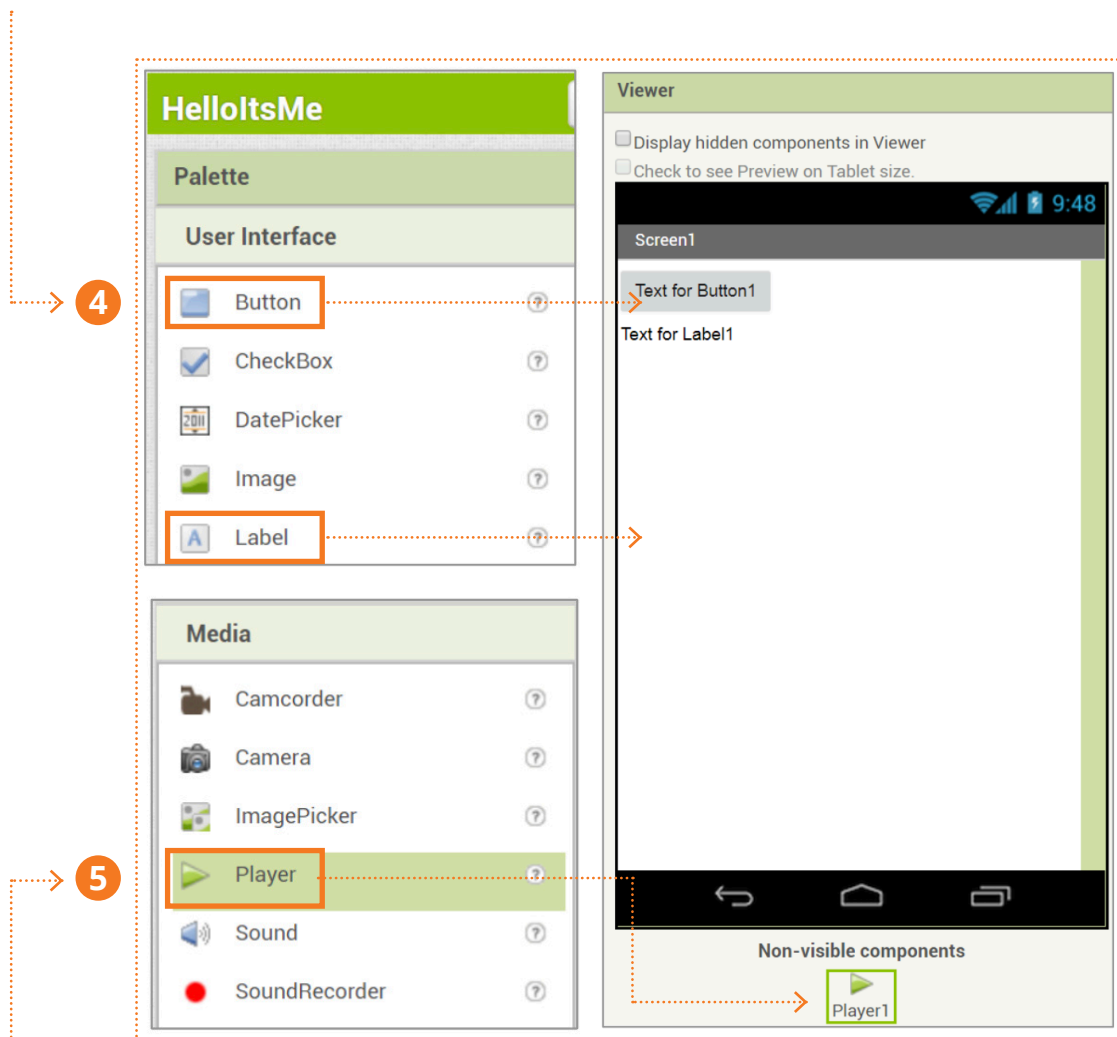


- ▶ 3 Name your project "HelloItsMe", then click **OK**.



## BUILD YOUR APP! .....

- ▶ **4** Add a **Button** and a **Label** component to your app by dragging them from the **User Interface** drawer in the Palette to the Viewer.



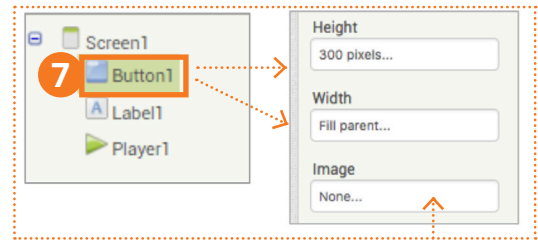
- ▶ **5** Add a **Player** component, a non-visible component, to your app by dragging it from the **Media** drawer.
- ▶ **6** Upload your picture and voice recording as media for the app.



## LET'S CONTINUE

▶ **7** Click on **Button1** in the Components window and change the button properties as follows:

- ▶ **Height:** 300 pixels
- ▶ **Width:** Fill parent



▶ **8** Find Image in the Properties window and click on **None**.

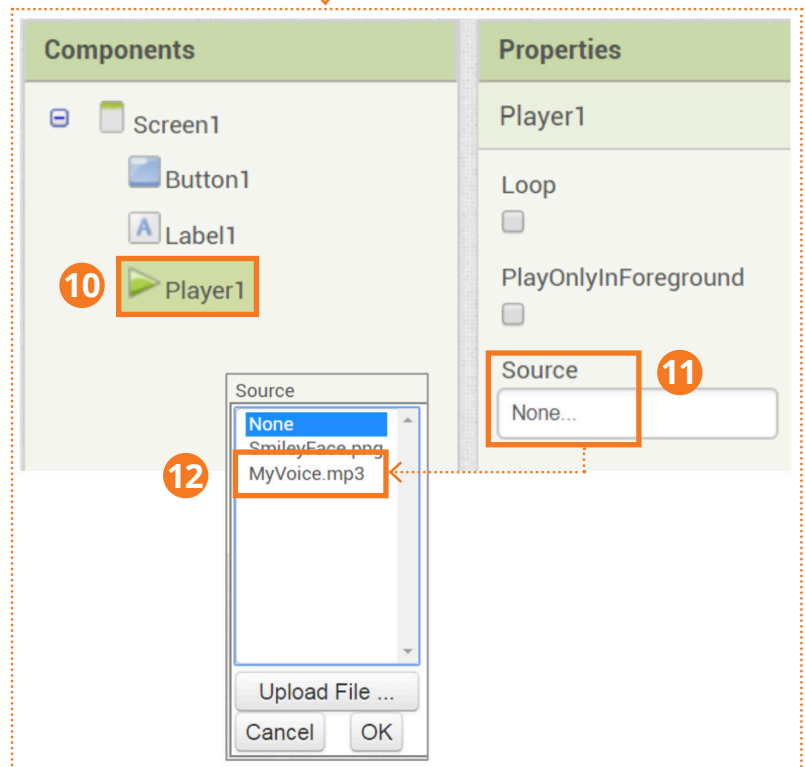
▶ **9** Click on the photo you uploaded earlier, then click **OK** to use it.



▶ **10** Choose **Player1** from the Components list.

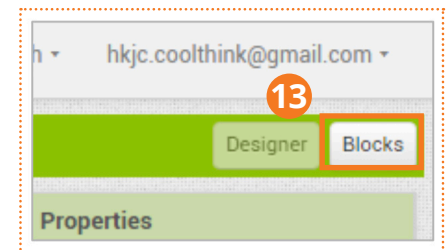
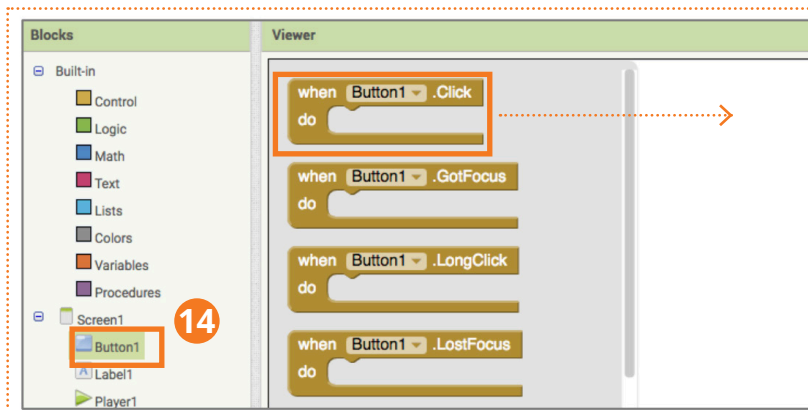
▶ **11** Find **Source** in the Properties window and click on **None**.

▶ **12** Click on the voice recording file you uploaded earlier, then click **OK** to use it.

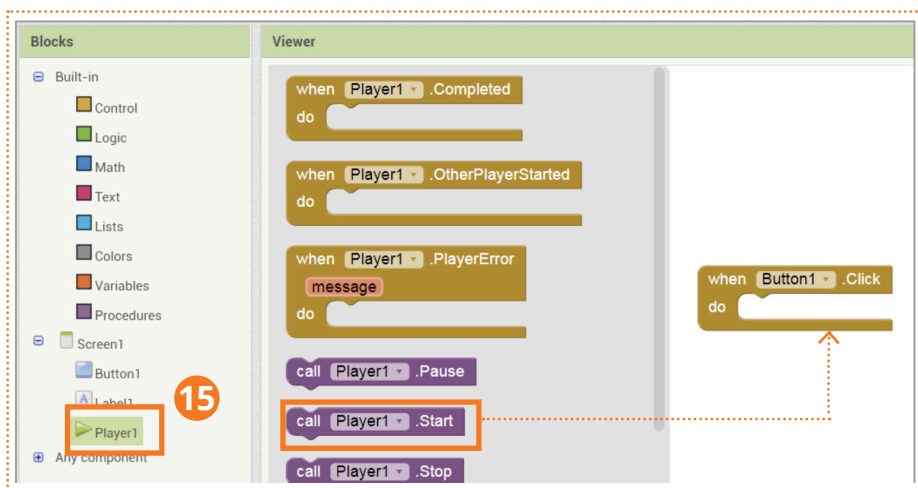


## BLOCKS EDITOR

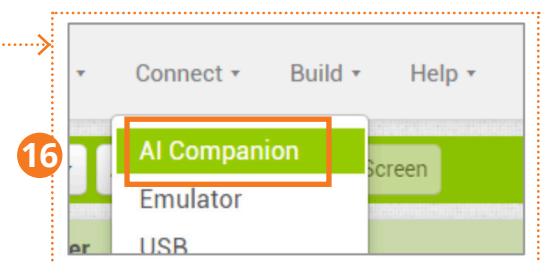
- ▶ **13** Click on the **Blocks** button and go to Blocks Editor.
- ▶ **14** Click on **Button1** in the Blocks component drawer, then drag out the **when Button1.Click** block.



- ▶ **15** Then click on **Player1** in the Blocks component drawer, and drag the **when Button1.Click** block out and snap it into the **when Button1.Click** block.



- ▶ **16** Now test your app on your smartphone or tablet with the MIT AI2 Companion app!



### CT Tips

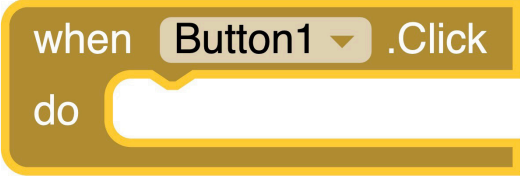
Remember what we learnt in the last lesson? The **when Button.Click** block is a typical **event**. It triggers actions.



## COMPUTATIONAL THINKING CONCEPTS

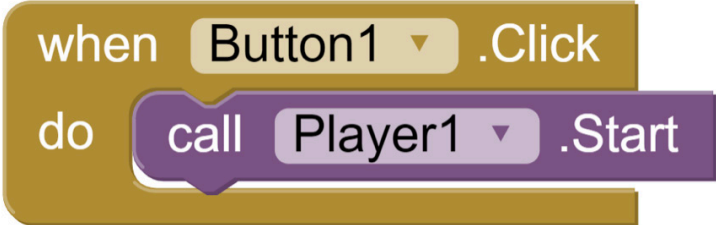
The following are the computational thinking concepts learnt in Lesson2.

### 1 Events



when **Button1** .Click  
do

### 2 Sequences



when **Button1** .Click  
do call **Player1** .Start

## COMPUTATIONAL THINKING PRACTICES

The following is the computational thinking practice used in this unit.

### 1 Testing and debugging:

- a) Test the changes to the app
- b) Fix any mistakes

## Lesson 1

## MY PIANO APP



In this activity, you will learn to reuse code in MIT App Inventor in order to build your own virtual piano!

## START HERE

- ▶ Go to the MIT App Inventor website and sign into your account. Open the **MyPiano\_template** project your teacher provides you..
- ▶ Most components have already been added to your piano app. See the components to the right.

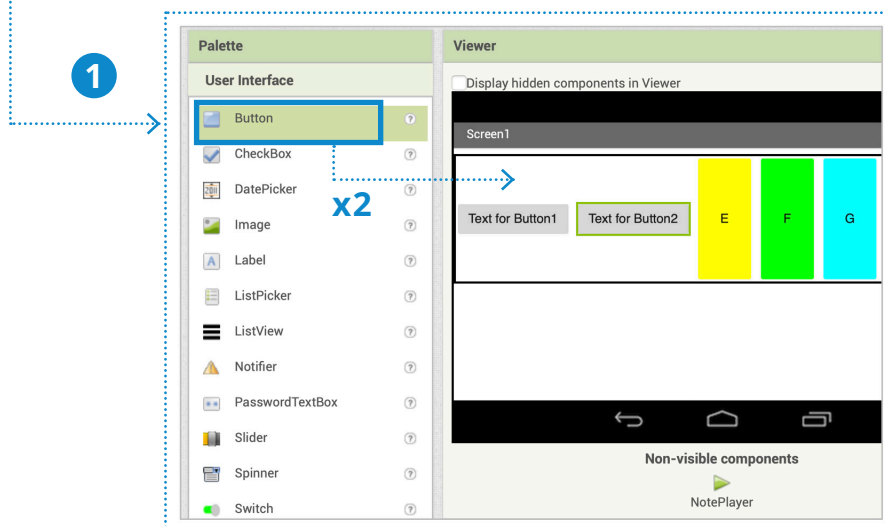
All the note buttons are named after the note. For example, "ENote", "FNote".



The **HorizontalArrangement1** lets you arrange the buttons horizontally.

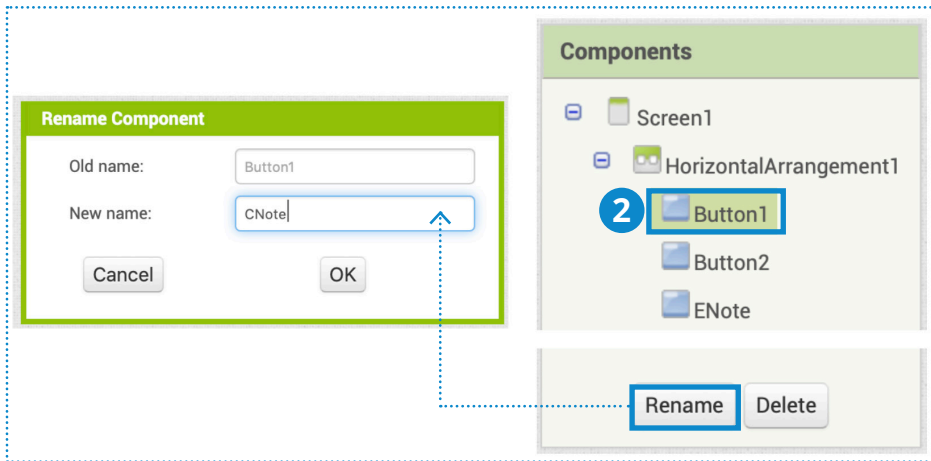
- ▶ **1** Drag two more Button components from the User Interface drawer into HorizontalArrangement1.

**non-visible component NotePlayer**



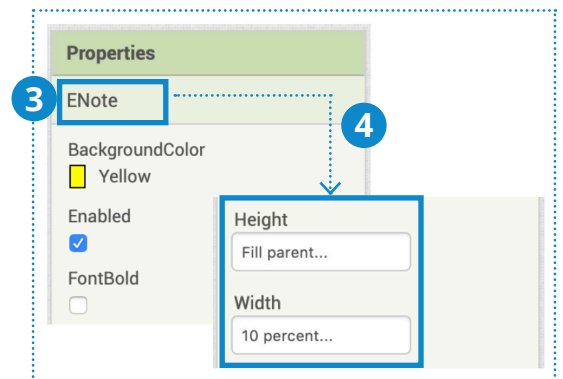
## CHANGE THE PROPERTIES

- ▶ **2** Rename **Button1** to “CNote” and **Button2** to “DNote”.

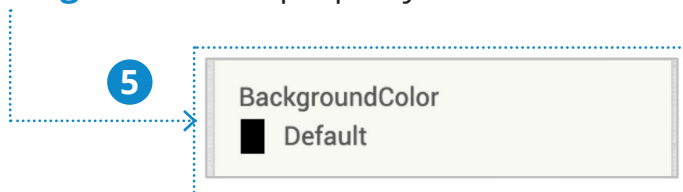


- ▶ **3** Click on **ENote** and look at **ENote** Properties.

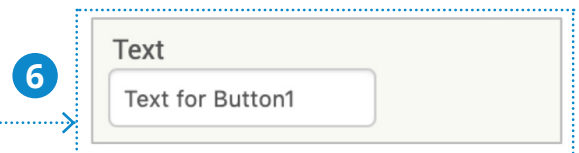
- ▶ **4** Change the **Height and Width** Properties of **CNote** and **DNote** buttons to match **ENote**.



- ▶ **5** Change the **BackgroundColor** property of **CNote** to **Red**.  
Change the **BackgroundColor** property of **DNote** to **Orange**.



- ▶ **6** Change the **Text** property of **CNote** to **C**.  
Change the **Text** property of **DNote** to **D**.



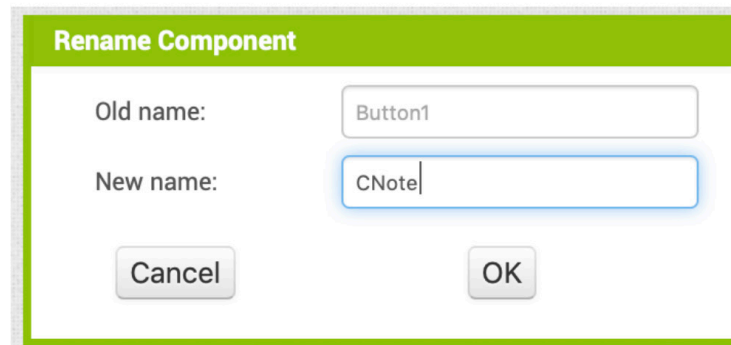
CT Tips

**Naming** components helps us to identify their functions. For example, keeping track of which button plays which note.

## COMPUTATIONAL THINKING CONCEPTS

The following is the computational thinking concept learnt in Lesson1.

### 1 Naming



**Rename Component**

Old name:

New name:

**Lesson 2**

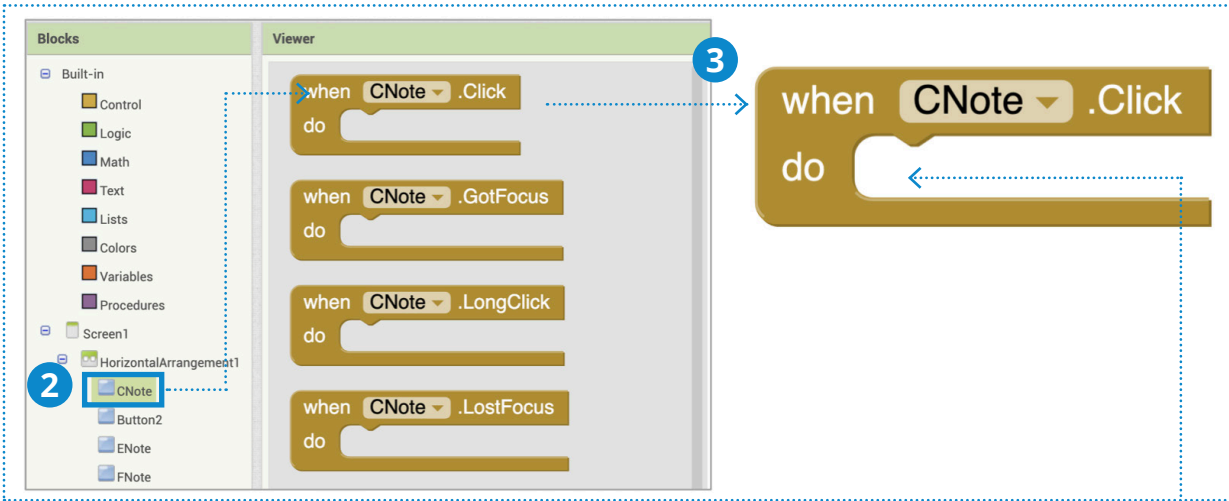
# MY PIANO APP

## CODE THE BUTTONS

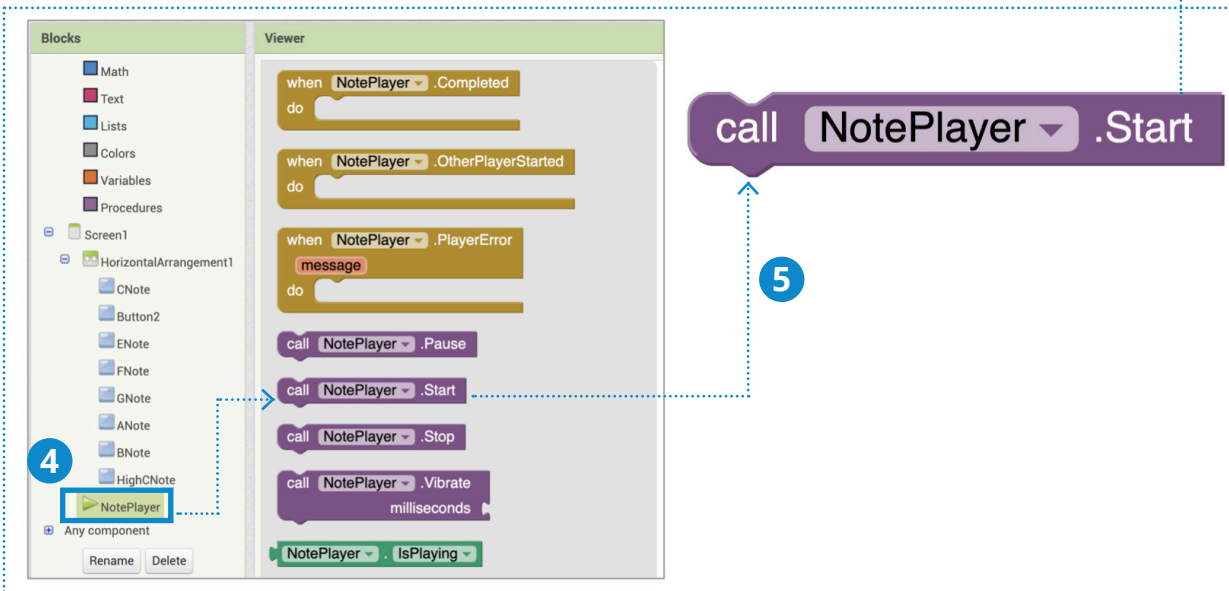
▶ **1** Open your MyPiano project in MIT App Inventor and go to the **Blocks Editor**.



▶ **2** **3** Click on **CNote** and drag the **when CNote.Click** block out.



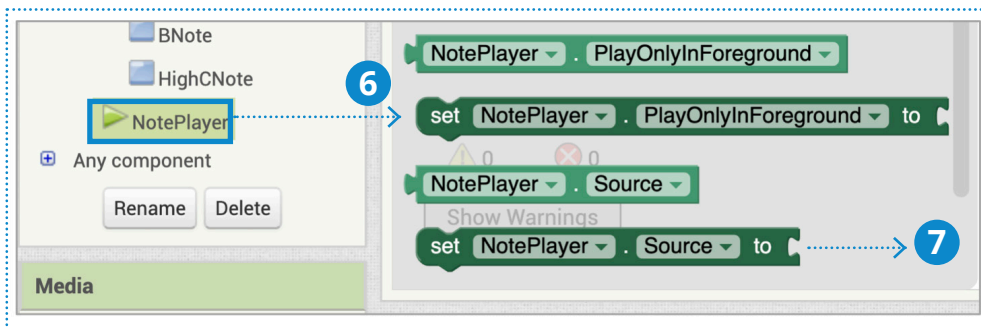
▶ **4** **5** Click on **NotePlayer**, drag the **call NotePlayer.Start** block out and snap it into the **when CNote.Click** block.



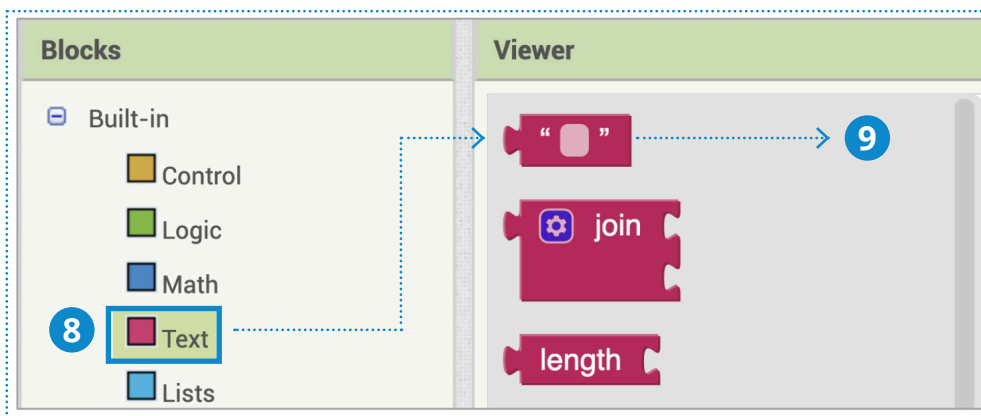
## LET'S CONTINUE .....

Since we will use the same Player component to play all the different notes, when we press a note button, let's set the Source to the proper.wav file.

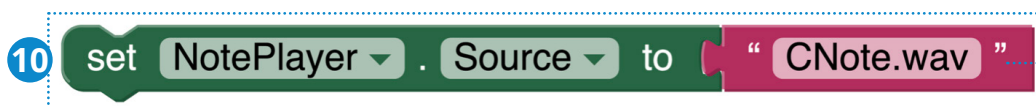
- ▶ **6 7** Drag out a **set NotePlayer.Source to** block.



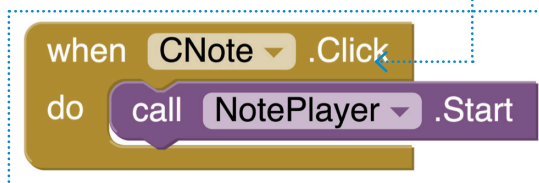
- ▶ **8 9** Drag out a blank text string block from the **Text** drawer.



- ▶ **10** Type "CNote.wav" into the blank text block and snap it into the **set NotePlayer.Source to** block.



- ▶ **11** Snap this block before the **call NotePlayer.Start** block.



CT Tips

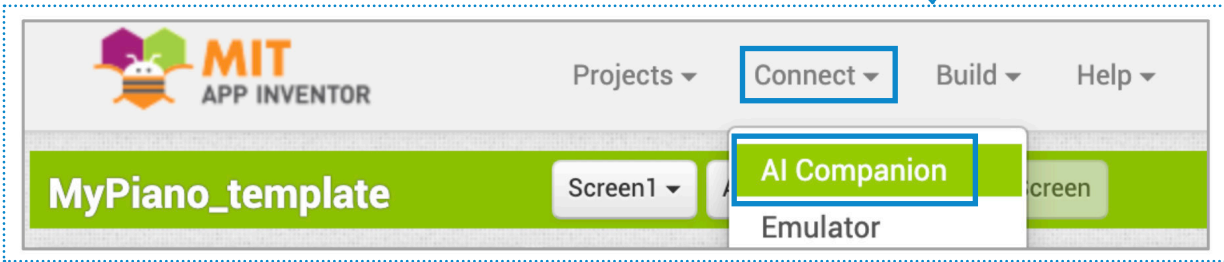
**when CNote.Click** is an **event** that triggers an action (e.g. play a sound file).

## LET'S CONTINUE

- ▶ **12** Test your app with the MIT AI2 Companion app. Press the **C** button to hear the **CNote** played.

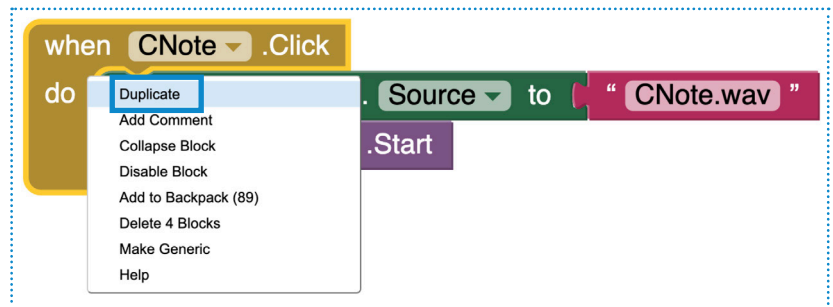
Since the **D** button will work like the C button, let's Duplicate the when **CNote.Click** block to use again.

12



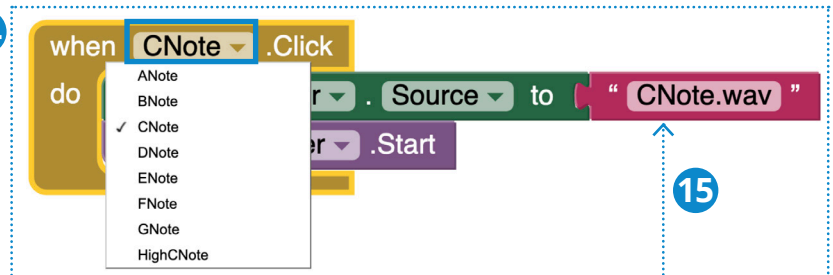
- ▶ **13** Right-click on the **when CNote.Click** block, and choose "Duplicate" from the dropdown menu.

13



- ▶ **14** Change **CNote.Click** to **DNote.Click** by using the dropdown menu in the duplicated block.

14

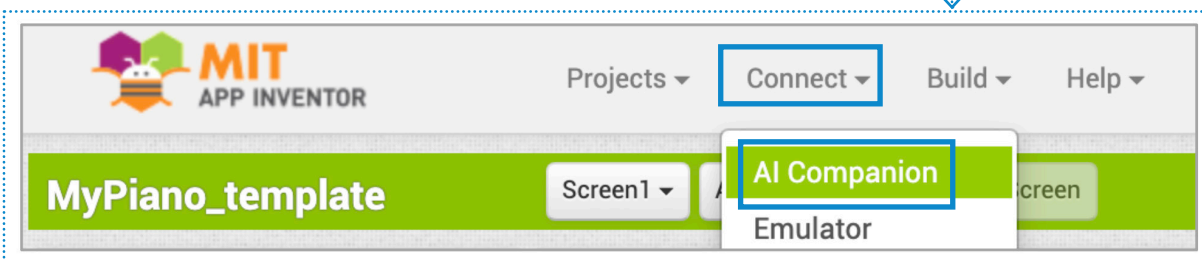


- ▶ **15** Change the text block from **Cnote.wav** to **DNote.wav**.

15

- ▶ **16** Test again with the MIT AI2 Companion app to see if both the **CNote** and **DNote** are played correctly

16

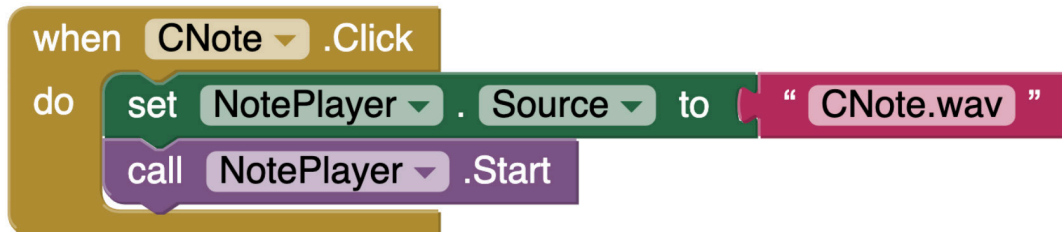


- ▶ **17** Can you code the remaining note buttons? Try it and test again with the MIT AI2 companion app.

## COMPUTATIONAL THINKING CONCEPTS

The following are the computational thinking concepts learnt in Lesson 2.

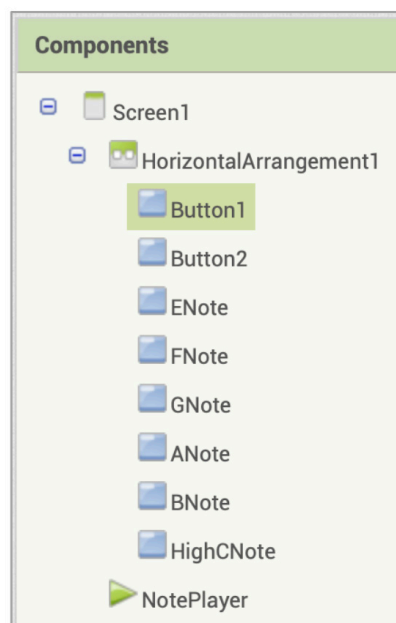
### 1 Sequences



### 2 Events



### 3 Naming





## COMPUTATIONAL THINKING PRACTICES

The following are the computational thinking practices used in this unit.

### 1 Reusing and remixing:

- a) Reuse and remix the code (playing sound) from the last unit
- b) Reuse and remix the code by copying/pasting the blocks

### 2 Being incremental and iterative:

- a) Add a note or two at a time
- b) Stop and test to make sure the new notes work
- c) Add more notes to make the app more interesting

### 3 Testing and debugging:

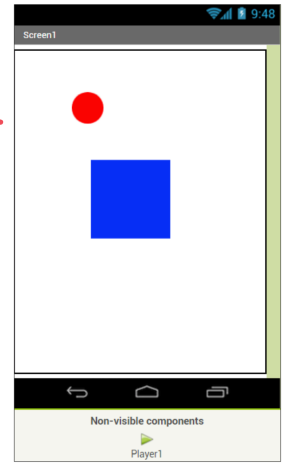
- a) Test that each note plays the right sound

**Lesson 1**

**FIND THE GOLD APP**

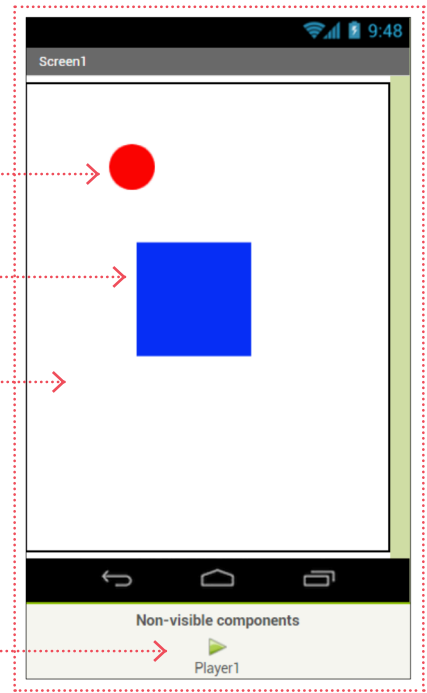
**START HERE**

The **SpriteCollide** app will give you some practice with the **Canvas**, **Ball** and **ImageSprite** components.



**SPRITE COLLIDE**

- ▶ Open the **SpriteCollide** app that your teacher will provide for you in MIT App Inventor.
- ▶ Look at the components included in the Designer.
  - ▶ **Ball1** moves around the screen randomly.
  - ▶ **SquareSprite** does not move, but we'll make the ball do something when they collide.
  - ▶ **Canvas1** is where the **Ball1** and **SquareSprite** are placed and can move.
  - ▶ **Player1** will play a sound when the **Ball1** and **SquareSprite** collide.
  - ▶ Now go to the **Blocks** Editor, and check out the blocks included so far in the app.



- ▶ **resetBall** is a procedure that resets some of **Ball1**'s properties.

- ▶ Colour to grey
- ▶ X,Y position to 50,50
- ▶ Speed to 10
- ▶ Heading (direction) to a random number

```

to resetBall
do
  set Ball1 . PaintColor to grey
  set Ball1 . X to 50
  set Ball1 . Y to 50
  set Ball1 . Speed to 10
  set Ball1 . Heading to random integer from 10 to 85
    
```

## LET'S CONTINUE

- ▶ The **when Screen1.Initialize** do block sets the position of SquareSprite to the center of the screen and calls the **call resetBall** procedure.

```

when Screen1.Initialize
do
  set SquareSprite.X to (Screen1.Width / 2) - 50
  set SquareSprite.Y to (Screen1.Height / 2) - 50
  call resetBall
    
```

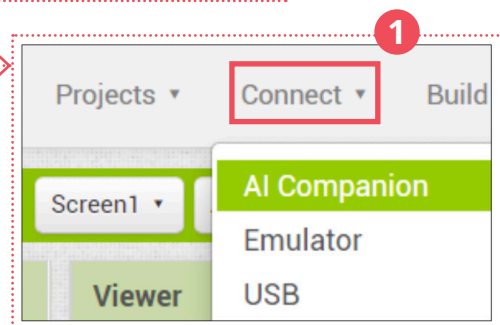
- ▶ **when Ball1.EdgeReached** do block is an event triggered when the Ball1 touches the edge of the screen.

```

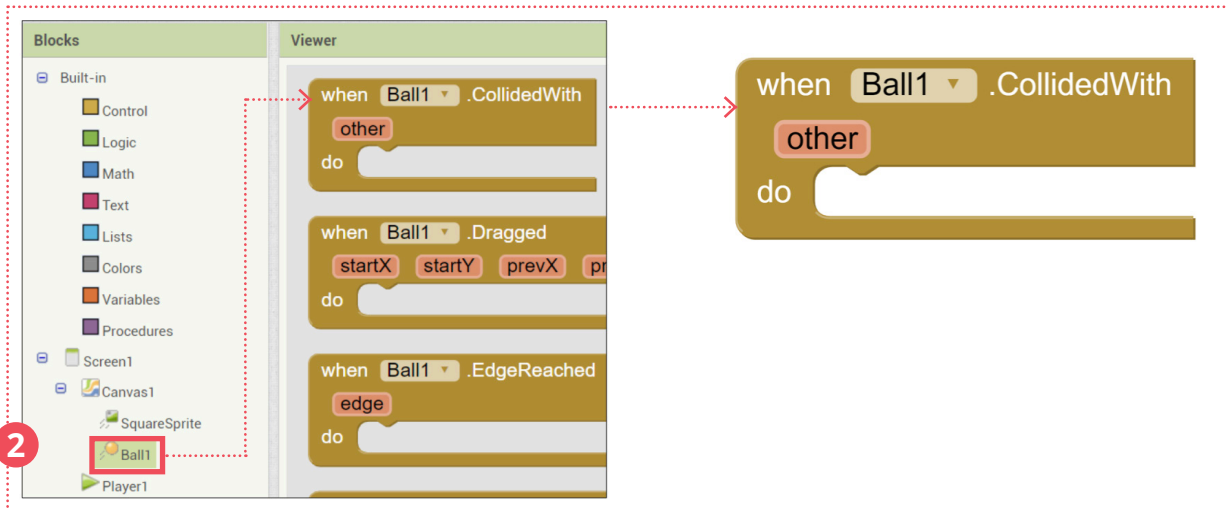
when Ball1.EdgeReached
  edge
do
  call Ball1.Bounce edge
  set Ball1.PaintColor to aqua
    
```

Ball1 bounces off the edge it touches...  
...and the colour changes to aqua.

- ▶ **1** Run the app using the MIT AI2 Companion app to see what happens!

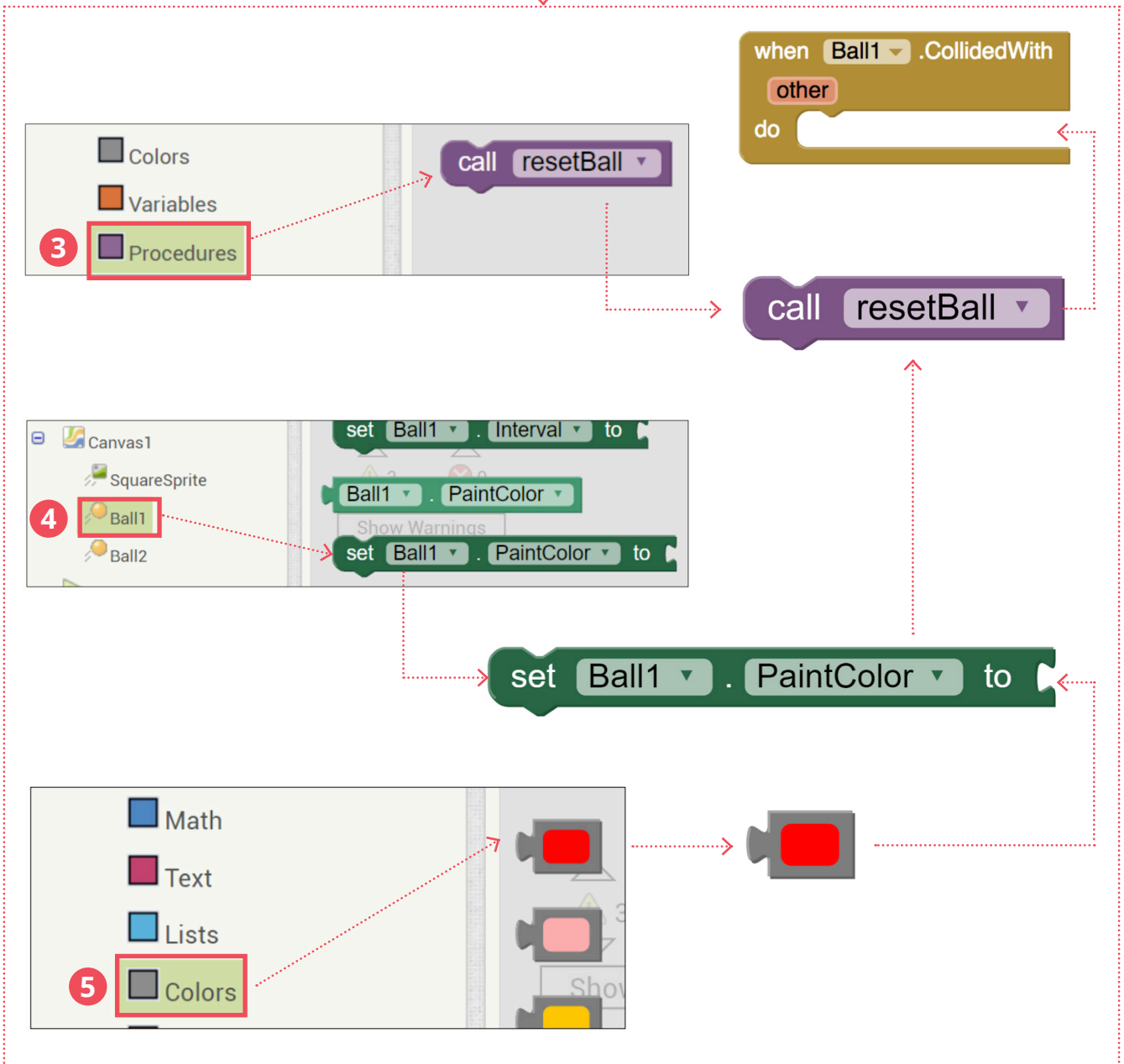


- ▶ **2** Drag out a **when Ball1.CollidedWith** do block.



## CHANGE BALL COLOUR

▶ Let's set Ball1 to change its colour and go back to its original position when it collides with SquareSprite.



CT Tips

**Event:** An event triggers actions. In this case, when the ball collides with something, it triggers certain actions.

## PLAY SOUND

► We can also add a sound effect by using the **set Player1.Source** block.

**6** Player1  
Any component

when Ball1 .CollidedWith  
other  
do  
call resetBall  
set Ball1 . PaintColor to

set Player1 . PlayOnlyInForeground to  
Player1 . Source  
set Player1 . Source to

set Player1 . Source to

**7** " crash.mp3 "  
" ding.wav "  
" fail.mp3 "

Choose one of the 3 sound files to play.

**8** Canvas1  
SquareSprite  
Ball1  
Ball2  
Player1  
Any component

call Player1 .Pause  
call Player1 .Start  
call Player1 .Stop  
call Player1 .Vibrate

call Player1 .Start

► **9** Test the app to see what happens when the ball and blue square collide!

Projects ▾ Connect ▾ Build

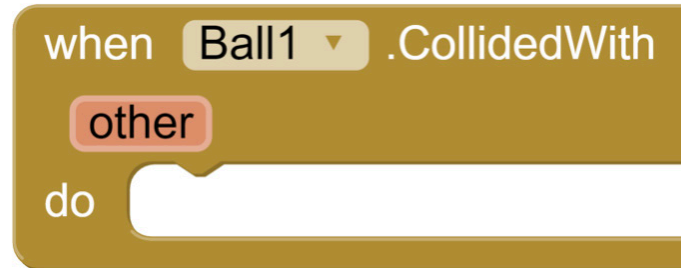
Screen1 ▾ AI Companion  
Emulator  
USB

Viewer

## COMPUTATIONAL THINKING CONCEPTS

The following are the computational thinking concepts learnt in Lesson 1.

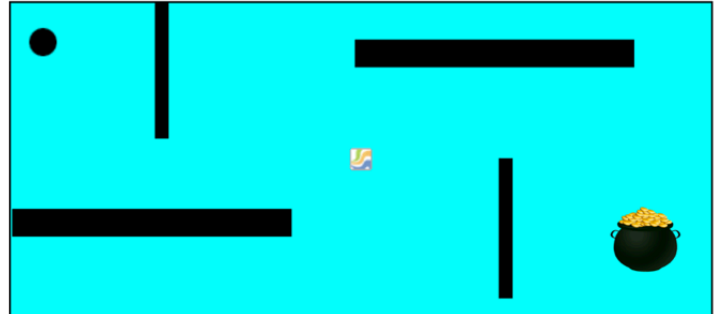
### 1 Events



**Lesson 2**

# FIND THE GOLD APP

In this unit, you will make a new maze game app that moves the ball through the maze by the tilting action on a smartphone or tablet.



**START HERE**

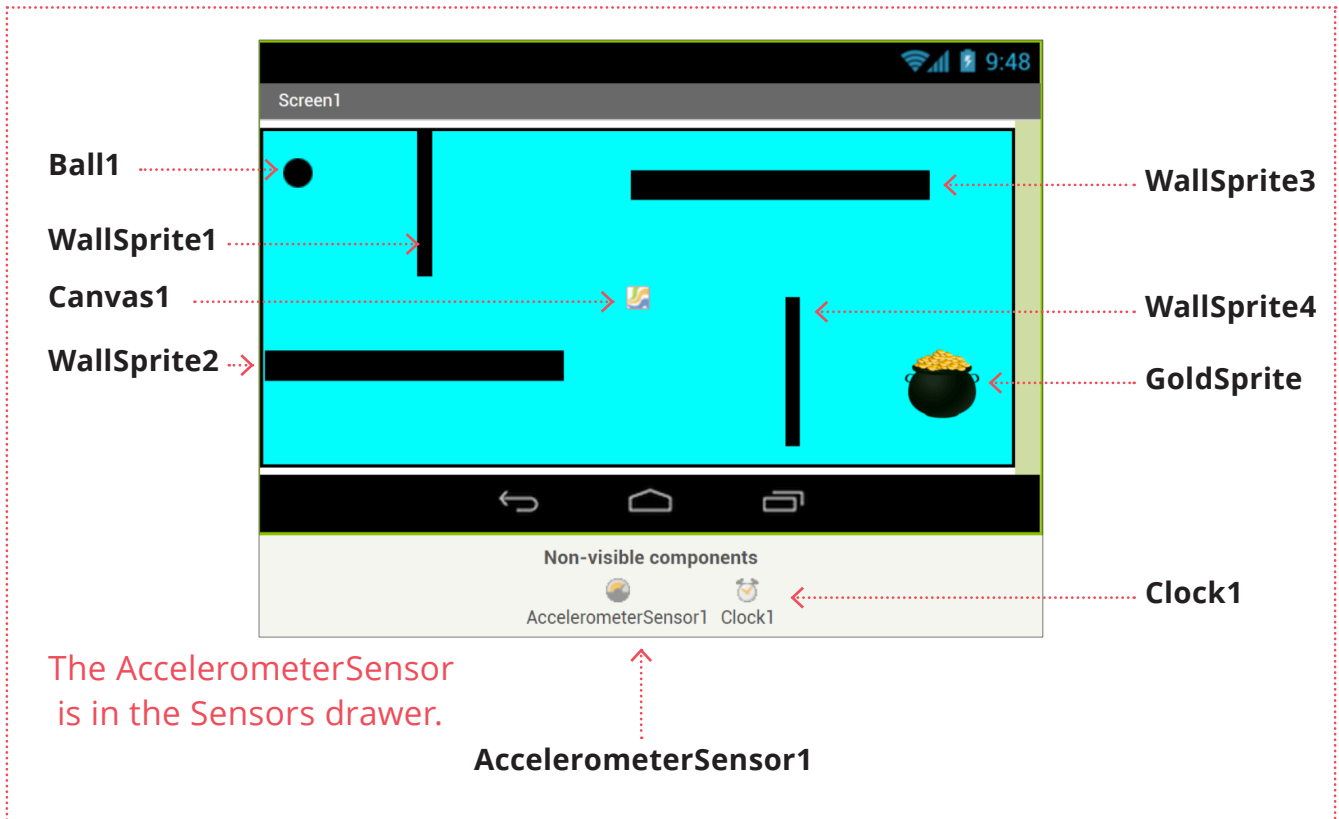
► Look at the following diagram with your partner, discuss, and try to complete the missing steps for the sequences for this app (some have been filled in for you).

A. When ball collides with...	B. the black wall	C. the Gold Pot
-------------------------------	-------------------	-----------------



## CONTINUE HERE

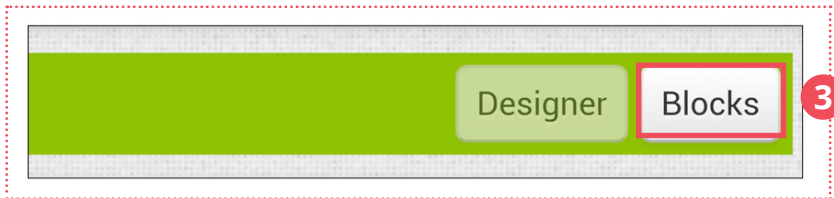
- ▶ **1** Open the **FindTheGold\_template.aia project** provided by your teacher in MIT App Inventor. This is what it should look like in the Designer.
- ▶ **2** Discuss with your partner the functions of each of the components.



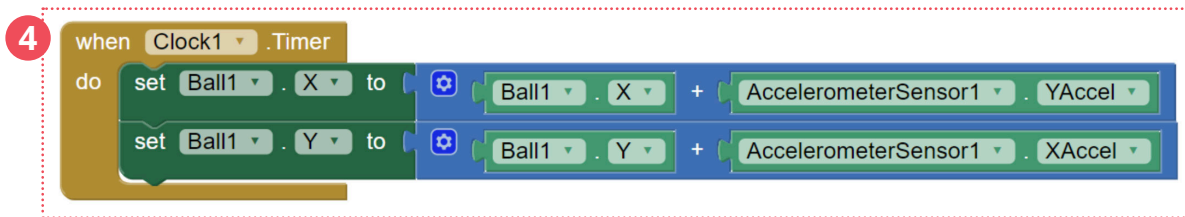


## CONTINUE HERE

- ▶ **3** Switch to the **Blocks** Editor.



- ▶ **4** We will use the **(when Clock1.Timer do block)** to update the X and Y positions of Ball1. The blocks have been completed for you.



## WHEN BALL COLLIDES WITH .....

- **5** **6** **7** Now we will check when the ball collides with the walls or the Gold-Sprite using the **when Ball1.CollidedWith do** block.

The image illustrates the step-by-step construction of a Scratch script:

- Step 5:** In the 'Blocks' palette, 'Ball1' is selected in the 'CollidedWith' dropdown of a 'when Ball1.CollidedWith do' block. This block is then placed into the 'Viewer' workspace.
- Step 6:** In the 'Built-in' palette, an 'if then' block is selected from the 'Control' category and placed into the 'do' area of the 'when Ball1.CollidedWith do' block.
- Step 7:** In the 'Built-in' palette, an '=' block is selected from the 'Logic' category and placed into the 'if' area of the 'if then' block.

The final assembled script block is shown as: **when Ball1.CollidedWith** (if then) (=)

## LET'S CONTINUE

- ▶ **8** **9** Set up a condition if **Ball1** collides with the **GoldSprite**.

The image illustrates the implementation of a collision condition in Scratch. It is divided into three main parts:

- Code Block Diagram:** A 'when Ball1 .CollidedWith' block is shown. Inside its 'do' section, there is an 'if' block. The 'if' block's condition is 'other = GoldSprite'. A red box highlights the 'other' variable in the 'do' section, and another red box highlights the 'GoldSprite' variable in the 'if' condition. A red arrow points from the 'other' variable in the 'do' section to the 'get other' block above it.
- Scratch Interface:** The 'Blocks' palette on the left shows a tree view of the project's objects. 'GoldSprite' is highlighted with a red box and a red circle containing the number '9'. A red arrow points from this 'GoldSprite' block to the 'GoldSprite' variable in the 'if' condition of the code block above.
- Variable Selection:** A separate 'GoldSprite' variable selection block is shown with a red arrow pointing to the 'GoldSprite' variable in the 'if' condition.

## LET'S CONTINUE

- ▶ **10 11** If **Ball1** collides with the **GoldSprite**, the app stops the **Timer** which will stop Ball1 moving.

The image illustrates the implementation of a collision event listener in Scratch. The code is as follows:

```

when Ball1 .CollidedWith
  other
do
  if
    get other = GoldSprite
  then
    set Clock1 . TimerEnabled to false

```

The interface shows the 'Clock1' timer component selected in the 'Any component' palette (labeled 10). The 'Logic' category in the 'Built-in' palette is selected (labeled 11), and the 'false' value is chosen from the dropdown menu. Red dashed arrows indicate the flow of information: the 'get other' block in the code points to the 'Clock1' component, and the 'false' value from the palette points to the 'TimerEnabled' property in the code.

## LET'S CONTINUE

- ▶ **12 13 14 15** When **Ball1** collides with any ImageSprite, it moves to the original position.

The image shows a sequence of steps for implementing collision logic in MIT App Inventor:

- Code Block:** A 'when Ball1 .CollidedWith other' block containing an 'if' block. The 'if' block checks 'get other = GoldSprite'. If true, it executes 'set Clock1 . TimerEnabled to false'.
- Canvas1:** A 'call Ball1 .MoveTo' block is shown with 'x' and 'y' inputs. A red box highlights the 'Ball1' object in the canvas.
- Built-in Library:** The 'Math' category is selected, showing a '0' block. A red box highlights the 'Math' category.
- Logic:** A '10' block is shown, representing the value to be moved to.
- Assembly:** The '10' block is being connected to the 'y' input of the 'call Ball1 .MoveTo' block. A 'Duplicate' button is visible below the '10' block.

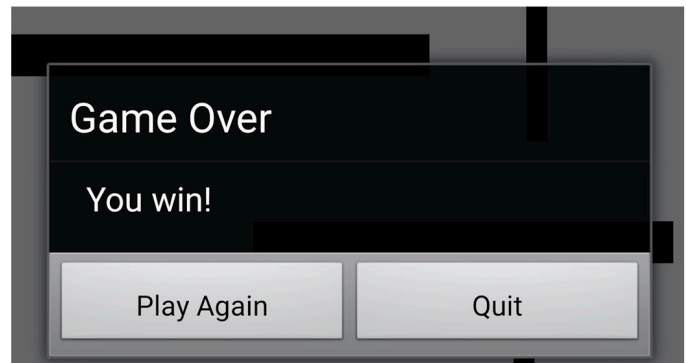
- ▶ **16** Test with the MIT AI2 Companion app.

The image shows the MIT AI2 Companion app interface with the following elements:

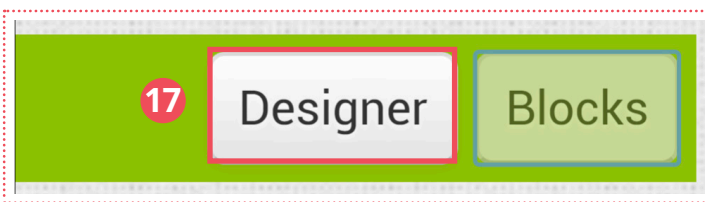
- Projects:** A dropdown menu showing 'Connect' (highlighted with a red box) and 'Build'.
- Screen1:** A dropdown menu showing 'AI Companion' (highlighted with a green box), 'Emulator', and 'USB'.
- Viewer:** A dropdown menu.
- A red box highlights the 'Connect' button and the 'AI Companion' option.

## NOTIFIER

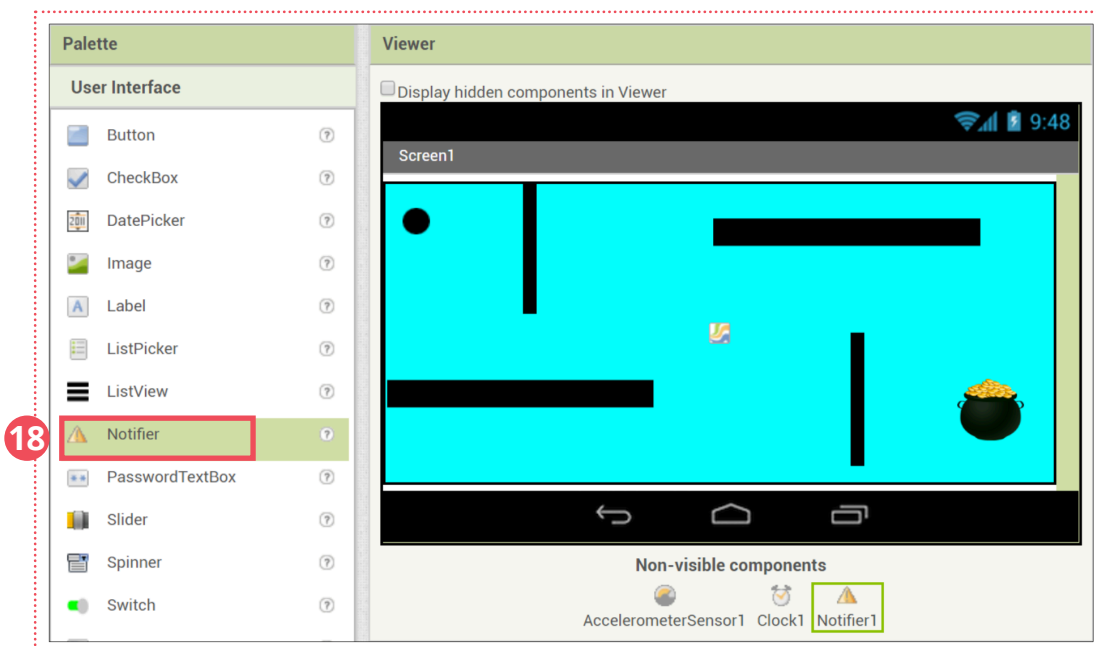
When the ball touches the gold sprite, let's notify the user the game is over and they can play again or quit.



- ▶ **17** Go to the **Designer**.



- ▶ **18** Let's add the **Notifier** component from **User Interface** drawer.

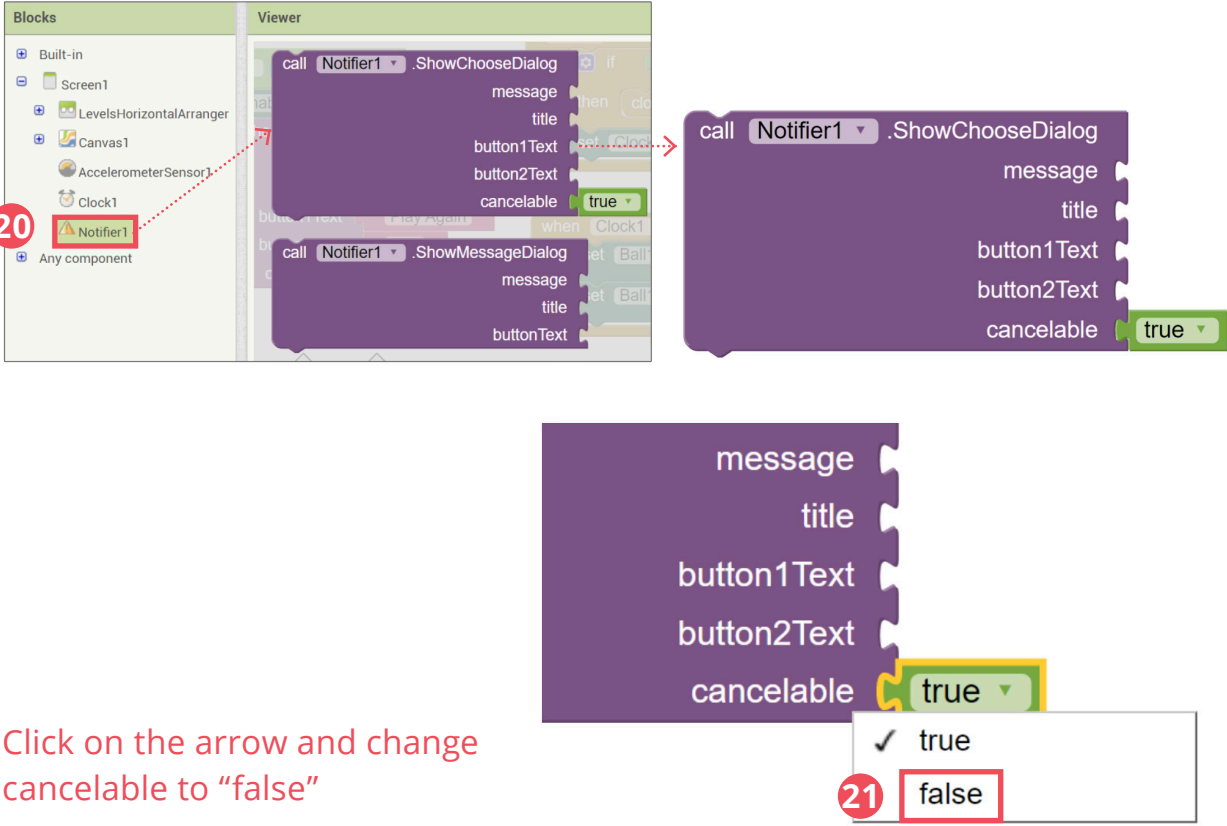


- ▶ **19** Go to **Blocks** Editor.



## LET'S CONTINUE

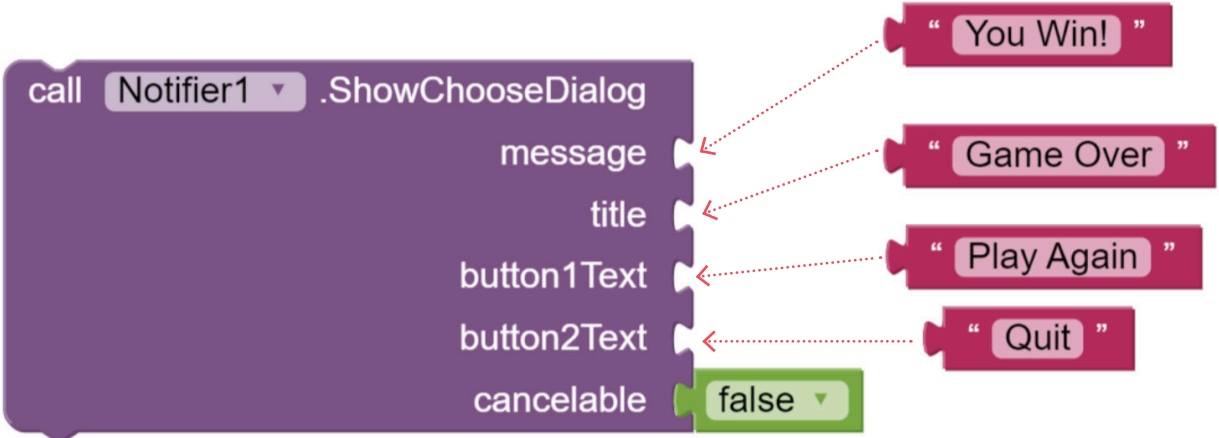
- **20** **21** Drag out a **call Notifier1.ShowChooseDialog...** block.



**20** Click on the arrow and change cancelable to "false"

**21** false

- **22** Drag the text to the corresponding slot.



**22** These blocks have been given to you.

## LET'S CONTINUE

- ▶ **24** **25** Drag the **call Notifier1.ShowChooseDialog...** block under the **set Clock1.TimerEnabled** to block so that the Notifier pops up when Ball1 collides with GoldSprite.

The image shows two parts of a Scratch script. The top part is a 'when Ball1 .CollidedWith other' block containing an 'if' block with 'get other = GoldSprite' and a 'then' block with 'set Clock1 .TimerEnabled to false'. Below this is a 'call Notifier1 .ShowChooseDialog' block with the following settings: message 'You Win!', title 'Game Over', button1Text 'Play Again', button2Text 'Quit', and cancelable 'false'. A red dashed arrow points from the 'set Clock1 .TimerEnabled to false' block to the 'call Notifier1 .ShowChooseDialog' block.

- ▶ **24** We use the **when Notifier1.AfterChoosing do** block to determine what to do when the user chooses a button.

The image shows the Scratch interface. On the left, the 'Blocks' palette has 'Notifier1' highlighted with a red box and labeled '24'. In the 'Viewer' area, a 'when Notifier1 .AfterChoosing choice' block is shown. A red dashed arrow points from this block to a larger version of the same block on the right. Below, the 'Built-in' palette has 'Control' highlighted with a red box and labeled '25'. A red dashed arrow points from 'Control' to an 'if then' block, which is then shown being placed into the 'do' slot of the 'when Notifier1 .AfterChoosing choice' block.



## LET'S CONTINUE

▶ 26 27 28 29 If user chooses "Quit", the app closes.

The image illustrates the process of adding a 'Quit' button and a 'close application' block to a Scratch script. It is divided into four numbered steps:

- Step 26:** The 'Logic' category is selected in the 'Built-in' section of the Blocks palette.
- Step 27:** A 'get choice' block is added to the 'if' statement's 'then' clause.
- Step 28:** A 'Quit' button is duplicated from the Stage. The text 'Right-click on your mouse and then click "Duplicate".' is shown next to the duplicated block.
- Step 29:** A 'close application' block is added to the script.

## LET'S CONTINUE

▶ **30 31** Enable the **Clock1.Timer**.

**30** when **Ball1** .CollidedWith  
 other  
 do  
 if **get other** = **GoldSprite**  
 then  
 set **Clock1** . **TimerEnabled** to **false**  
 call **Notifier1** .ShowChooseDialog  
 message  
 title  
 button1Text  
 button2Text  
 cancelable  
 call **Ball1** .MoveTo  
 x 10  
 y 10

**31** set **Clock1** . **TimerEnabled** to **true**

Right-click on your mouse and then click "Duplicate".

▶ **32** Finally, Test and Debug using the MIT AI2 Companion app.

**32** Projects **Connect** Build  
 Screen1 **AI Companion**  
 Emulator  
 Viewer USB

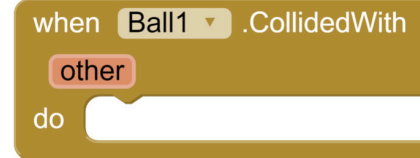
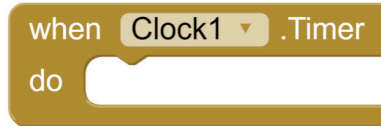
**CT Tips**

**Conditionals:** "if" a condition is met, "then" do something. In this case, "if" the user chooses to quit, "then" the app closes.

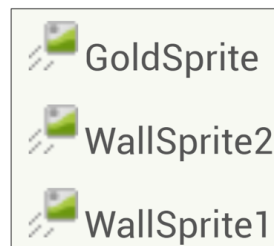
## COMPUTATIONAL THINKING CONCEPTS

The following are the computational thinking concepts learnt in Lesson 2.

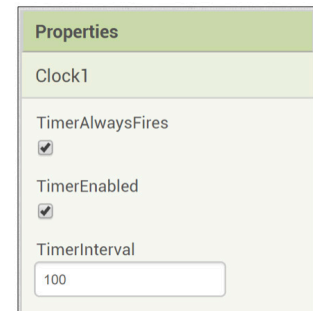
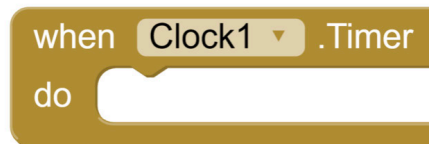
### 1 Events



### 2 Naming: ImageSprites



### 3 Repetition



### 4 Conditionals

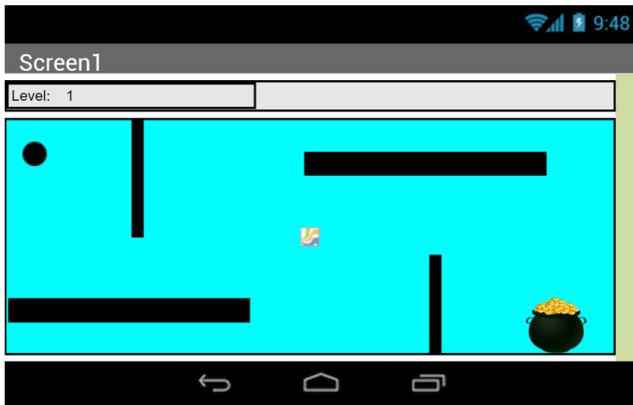
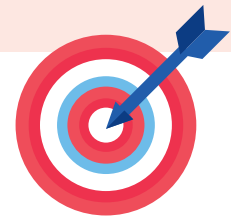


### 5 Operators



## Lesson 3

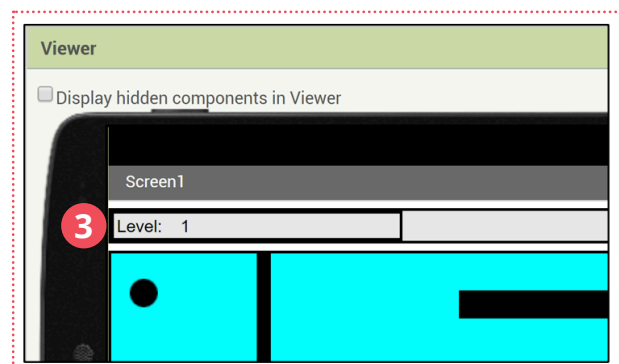
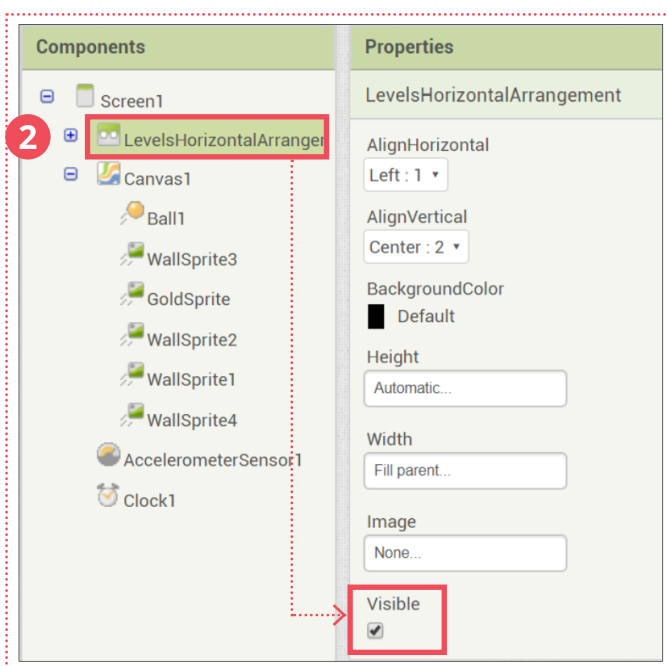
## FIND THE GOLD APP



In this lesson, you will add a new feature to make your app more interesting by increasing the level of difficulty!


**START HERE**

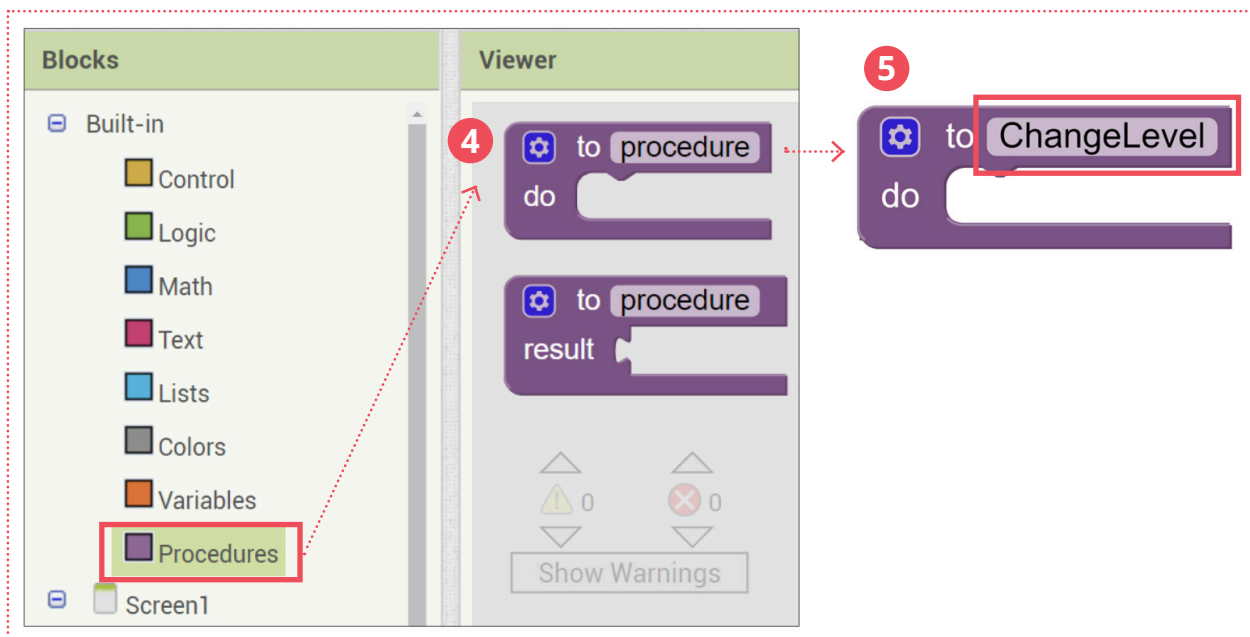
- ▶ **1** Open your **Find The Gold** project.
- ▶ **2** Look at the **Components** tab, click on the **LevelsHorizontalArrangements** and check the **Visible** box .
- ▶ **3** Then you will see **"Level: 1"** at the top-left of the app which has been completed for you.



## MAKING A PROCEDURE

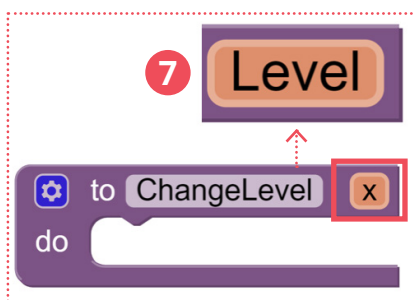
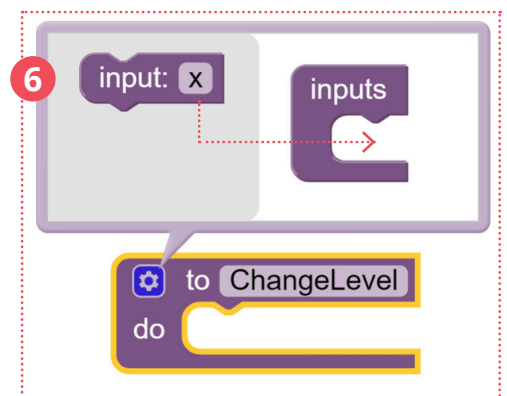
In order to increase the difficulty, we need to make the ball move faster when user reaches the gold. We will make a procedure to do it.

- ▶ **4** Drag out the **to procedure do** from the **Procedure** drawer.
- ▶ **5** Rename “procedure” to **“ChangeLevel”**.



Now we will add something new for the procedure, an input parameter.

- ▶ **6** Click on the blue gear of the **to ChangeLevel do** block and drag **“input:x”** inside the **inputs** block.
- ▶ **7** You now have an input parameter named “x”. Rename that “x” to **“Level”**.

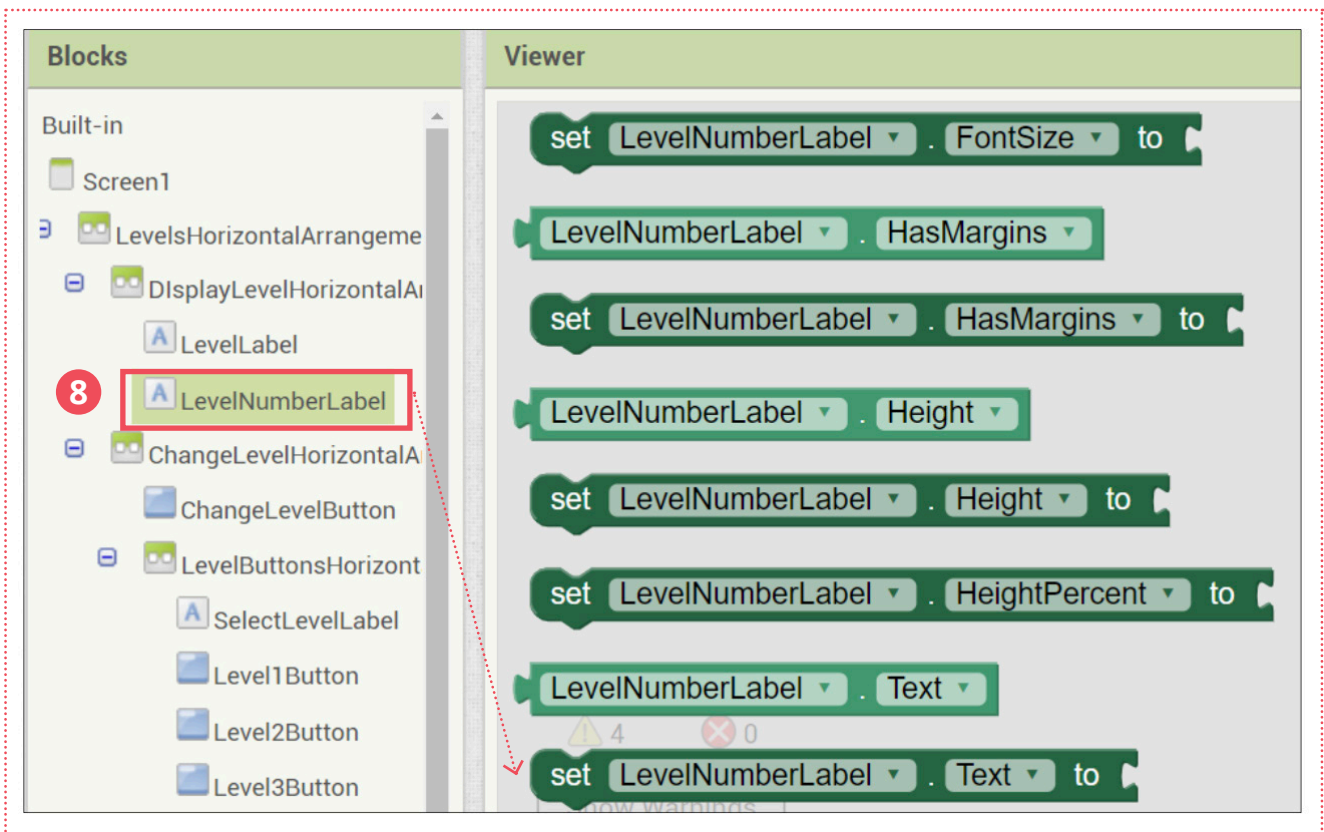


## LET'S CONTINUE

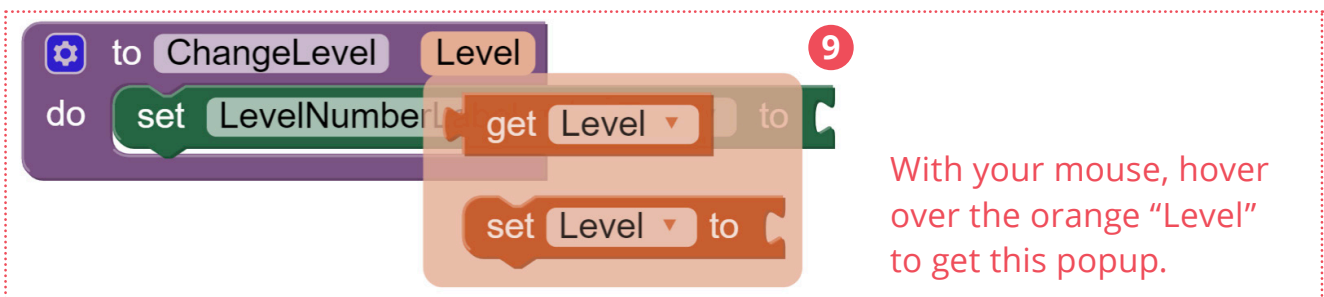
With the parameter, we can pass the procedure different values when we call it.

Let's build the **ChangeLevel** procedure. First, it will update the text of LevelNumberLabel.

- ▶ **8** Drag out a **set LevelNumberLabel.Text to** block and snap it into the procedure.



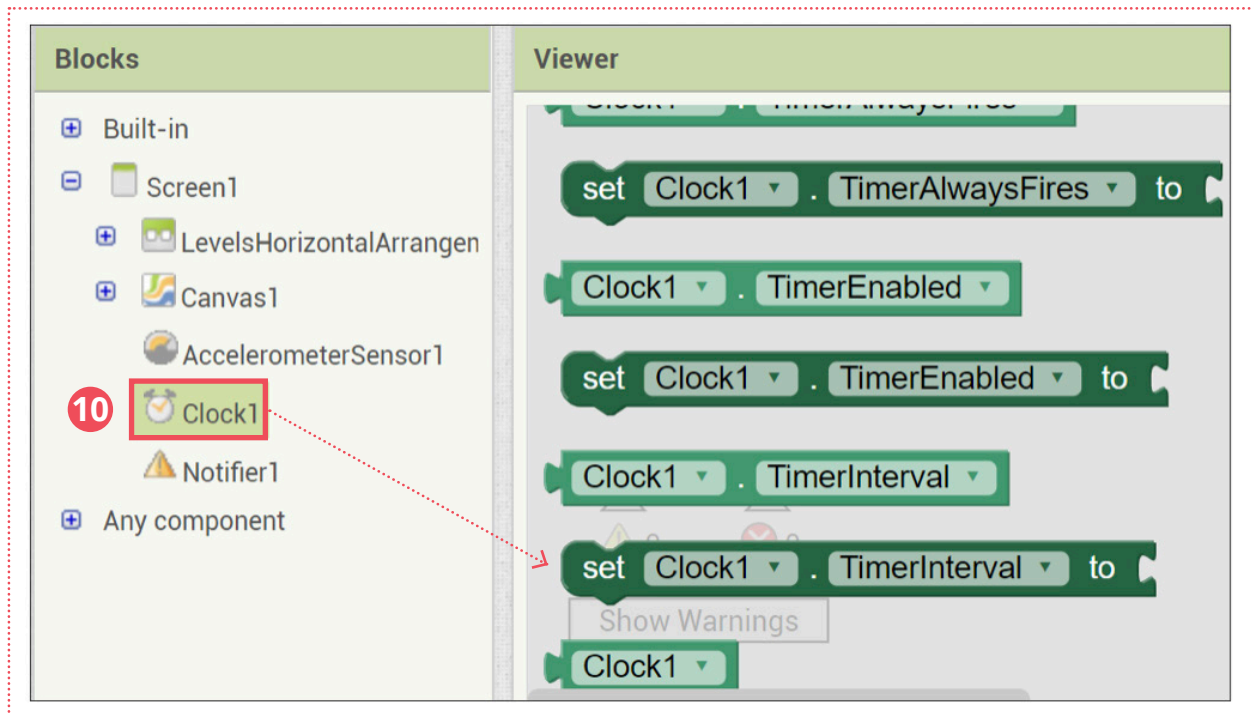
- ▶ Hover over the mouse on the parameter, Level.
- ▶ **9** Snap **get Level** to the **set LevelNumberLabel.Text to** block.



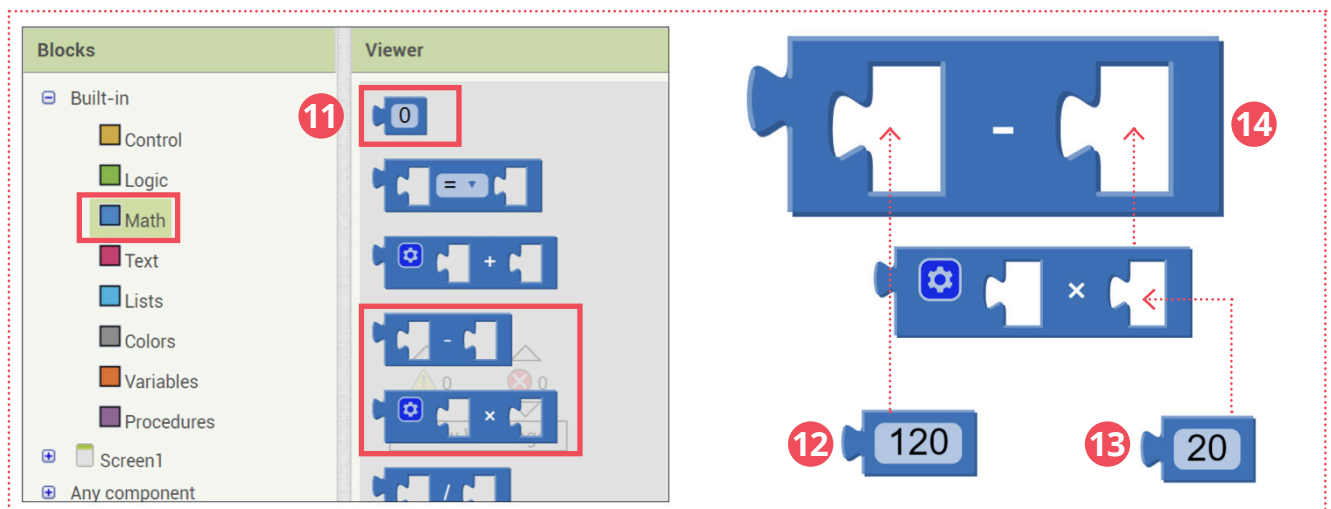
## LET'S CONTINUE

Secondly, the procedure will make the ball move faster depending on the level. We can do it by changing the Clock's TimerInterval.

- ▶ **10** Draw out a **set Clock1.TimerInterval to** block and snap it into the procedure.



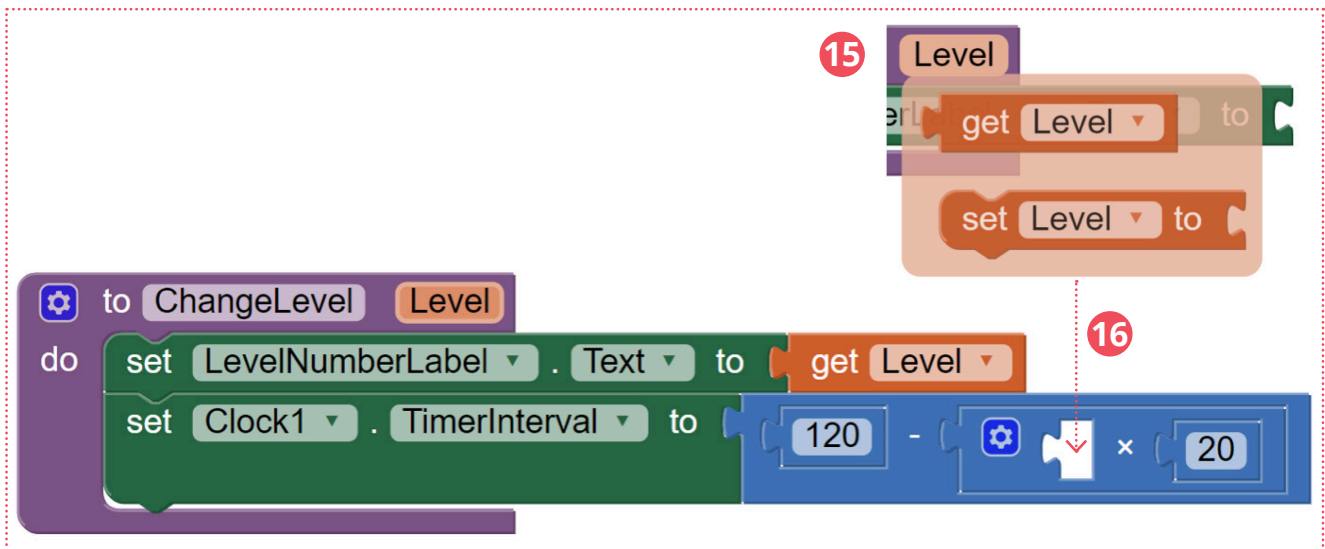
- ▶ **11** From the **Math** drawer, drag out a **- (subtract)** block, a **x (product)** block and two number blocks. Change the two numbers to **"120"** and **"20"**.



- ▶ **12** Snap the **"120"** block into the **first part of the - block**.
- ▶ **13** Snap the **"20"** block into the **second of part of the x block**.
- ▶ **14** Snap the **x** block into the **second of part of the - block**.

## LET'S CONTINUE .....

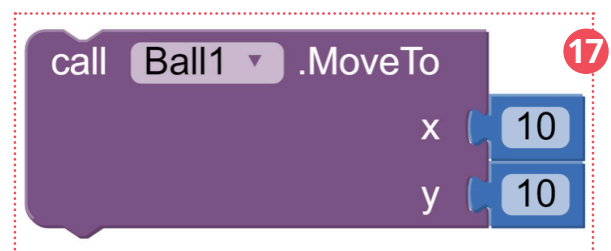
- ▶ Hover over the mouse again to get the **get level** block.
- ▶ **15** Snap the **get level** block into the first part of the x block.
- ▶ **16** Snap the whole - block to the **set Clock1.TimerInterval to** block in the procedure.



## BALL RETURN .....

Our procedure should have one more thing. We should also set a procedure where the ball returns to the starting position.

- ▶ **17** Do you remember how to move **Ball1** to its original position that we learnt in Lesson 2?



- ▶ **18** Try to make the above blocks and snap it into the procedure yourself.



### CT Tips

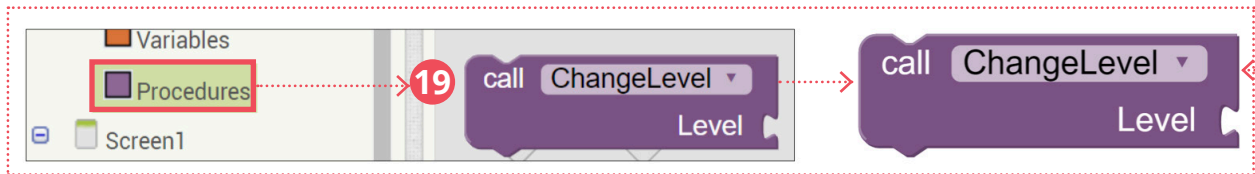
**Procedure:** You can package a block of code into a procedure and call it in different places in a program.



## CALL THE PROCEDURE

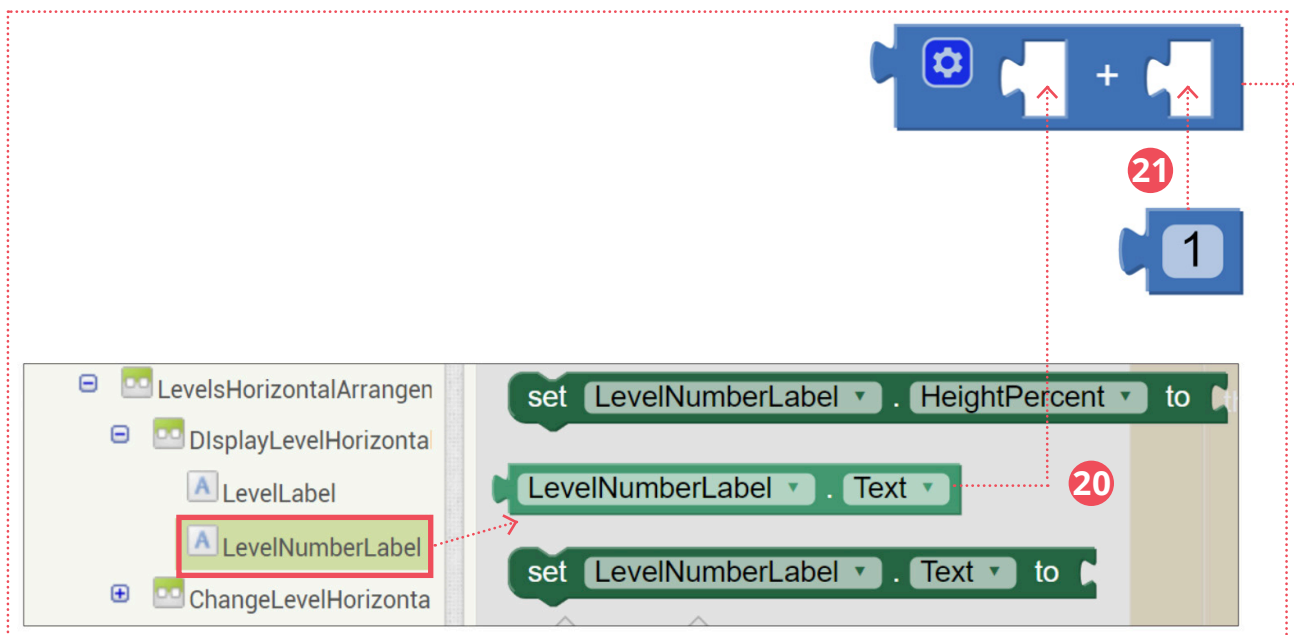
When the user reaches the Gold sprite, we will then **“ChangeLevel”**.

- ▶ **19** Draw out a **call ChangeLevel** block with **Level** parameter.



Also, we will increase the level by 1 every time we call the procedure.

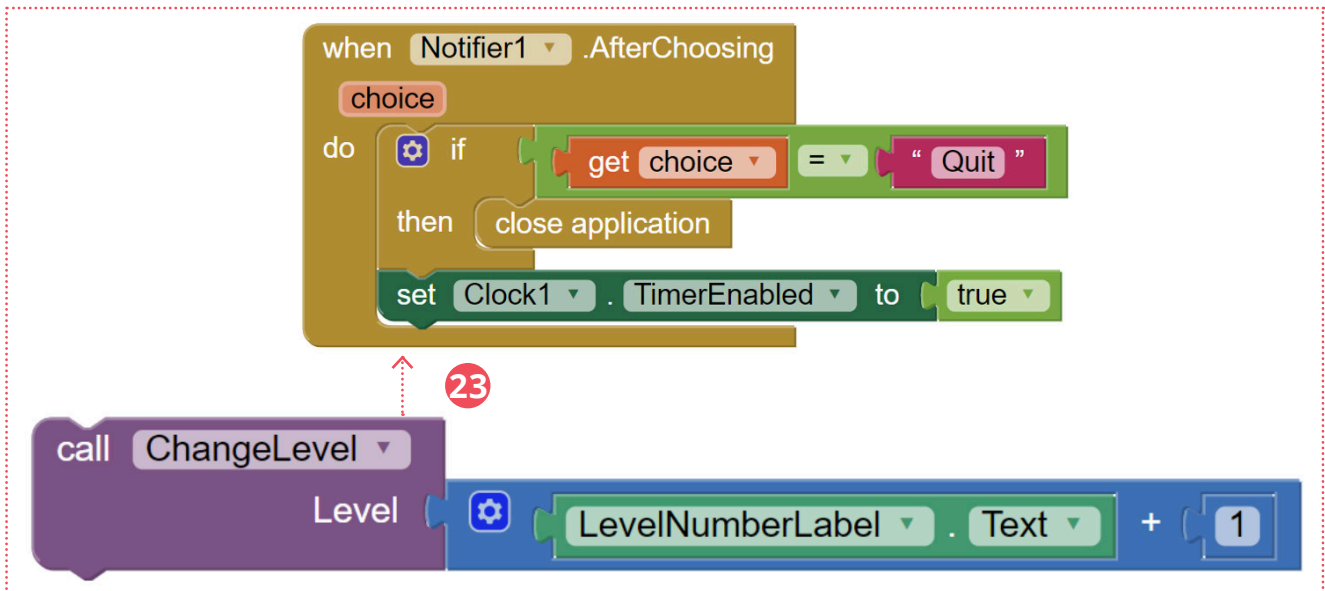
- ▶ Drag out a **+** (**plus**) block and a number block from the **Math** drawer. **22**
- ▶ **20** Drag out the **LevelNumberLabel.Text** block.
- ▶ Snap the **LevelNumberLabel.Text** into the **first part of the + block**.



- ▶ **21** Change the number to **“1”** and snap it into the **second part of the + block**.
- ▶ **22** Then snap the whole + block in the **Level** parameter in the **call ChangeLevel** block.

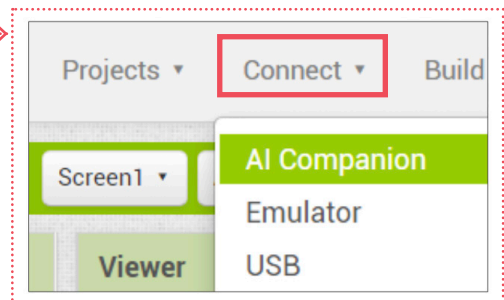
## LET'S CONTINUE

- ▶ **23** Finally, snap the **call ChangeLevel** block into the **when Notifier1.AfterChoosing do** block.



## TESTING & DEBUGGING

- ▶ **24** Well done! Now use the MIT AI2 Companion app to see what happens after you “find the gold”!



## COMPUTATIONAL THINKING CONCEPTS

The following are the computational thinking concepts learnt in Lesson 3.

### 1 Sequences

### 2 Naming

#### Procedure and Parameter

#### Level Labels

### 3 Repetition

## Make a Pizza with a Procedure's Instruction

Please distribute the printout of recipes to the students selected, and follow the instructions below to run the Unplugged Activity: Make a Pizza with a Procedure.

1. Ask four students to come up to the front of the room to represent four different pizza restaurants. One at a time, give each of them a card with a pizza recipe.

<p><b>1. MUSHROOM PIZZA RECIPE</b></p> <ul style="list-style-type: none"><li>• Roll out dough.</li><li>• Spread tomato sauce on dough.</li><li>• Sprinkle cheese on tomato sauce.</li><li>• Sprinkle sliced mushrooms on top of cheese.</li><li>• Put pizza in oven at 350oC for 5 minutes.</li><li>• Take out of oven.</li></ul>	<p><b>2. PEPPERONI PIZZA RECIPE</b></p> <ul style="list-style-type: none"><li>• Roll out dough.</li><li>• Spread tomato sauce on dough.</li><li>• Sprinkle cheese on tomato sauce.</li><li>• Sprinkle sliced pepperoni on top of cheese.</li><li>• Put pizza in oven at 350oC for 5 minutes.</li><li>• Take out of oven.</li></ul>
<p><b>3. PEPPER PIZZA RECIPE</b></p> <ul style="list-style-type: none"><li>• Roll out dough.</li><li>• Spread tomato sauce on dough.</li><li>• Sprinkle cheese on tomato sauce.</li><li>• Sprinkle sliced peppers on top of cheese.</li><li>• Put pizza in oven at 350oC for 5 minutes.</li><li>• Take out of oven.</li></ul>	<p><b>4. ANCHOVY PIZZA RECIPE</b></p> <ul style="list-style-type: none"><li>• Roll out dough.</li><li>• Spread tomato sauce on dough.</li><li>• Sprinkle cheese on tomato sauce.</li><li>• Sprinkle sliced anchovies on top of cheese.</li><li>• Put pizza in oven at 350oC for 5 minutes.</li><li>• Take out of oven.</li></ul>

2. Have the four students read aloud their own recipe.
3. Ask the class the following questions:
  - (1) Do you see anything similar among the different recipes?
  - (2) What is different?
  - (3) Could it be more efficient?

## Make a Pizza with a Procedure's Instruction

4. Present the solution by giving the fifth student a generic pizza recipe, and tell that student what kind of pizza you want and they can make it.
5. Have the fifth student come to the front of the class and read aloud the recipe of Restaurant 5.

### 5. PIZZA RECIPE WITH OPTIONAL INGREDIENT

- Roll out dough.
- Spread tomato sauce on dough.
- Sprinkle cheese on tomato sauce.
- Sprinkle sliced optional ingredient on top of cheese.
- Put pizza in oven at 350oC for 5 minutes.
- Take out of oven.

6. Ask students to compare these five pizza restaurants' recipes. Explain that the first four restaurants are not very versatile because they can only make one type of pizza. The last restaurant is a better solution because it is capable of making any kind of pizza, as long as we tell the restaurant what optional ingredient we want. If we want to update the pizza recipe (such as adjusting the amount of salt or adding new spices) for the restaurant, we just need to modify one copy.

## Make a Pizza with a Procedure's Instruction

7. (Optional) Demonstrate the versatility of the new student with the single recipe (only include this step if there is enough time).
  - (1) Teacher approaches student 1 in the line and says "I want one mushroom and one anchovy pizza."
  - (2) Student 1 responds "I can give you the mushroom pizza, but that's all I know how to make".
  - (3) Teacher goes to Student 2 and asks for the two different pizza. Student 2 responds "Sorry, we don't make those kinds of pizza".
  - (4) Teacher goes to Student 3 and asks the same. Same response as Student 2.
  - (5) Teacher goes to Student 4 and asks the same. Student 4 responds "I can give you the anchovy pizza, but we don't make mushroom pizzas".
  - (6) Then teacher goes to Student 5 with the generic recipe and asks the same. Student 5 responds "Sure, we make those two pizzas. First I will make the mushroom pizza. Then I will make the anchovy pizza"
  - (7) Teacher can explain that he/she could have gotten the two pizza from the other four students but he had to go down the line to find the one who sold those types of pizza. The last student/restaurant is more versatile and helpful, because he was able to produce what the teacher wanted.

## Make a Pizza with a Procedure's Instruction

### 1. MUSHROOM PIZZA RECIPE

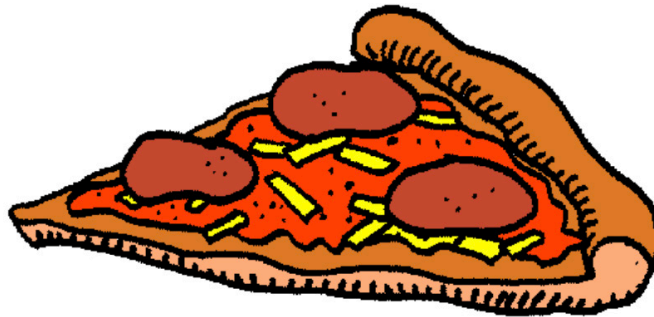
- Roll out dough.
- Spread tomato sauce on dough.
- Sprinkle cheese on tomato sauce.
- Sprinkle sliced mushrooms on top of cheese.
- Put pizza in oven at 350oC for 5 minutes.
- Take out of oven.



## Make a Pizza with a Procedure's Instruction

### 2. PEPPERONI PIZZA RECIPE

- Roll out dough.
- Spread tomato sauce on dough.
- Sprinkle cheese on tomato sauce.
- Sprinkle sliced pepperoni on top of cheese.
- Put pizza in oven at 350°C for 5 minutes.
- Take out of oven.

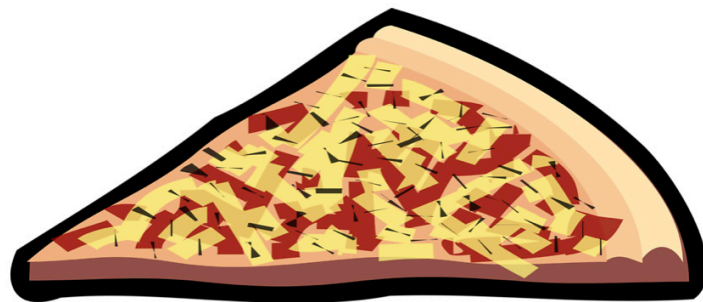




## Make a Pizza with a Procedure's Instruction

### 3. PEPPER PIZZA RECIPE

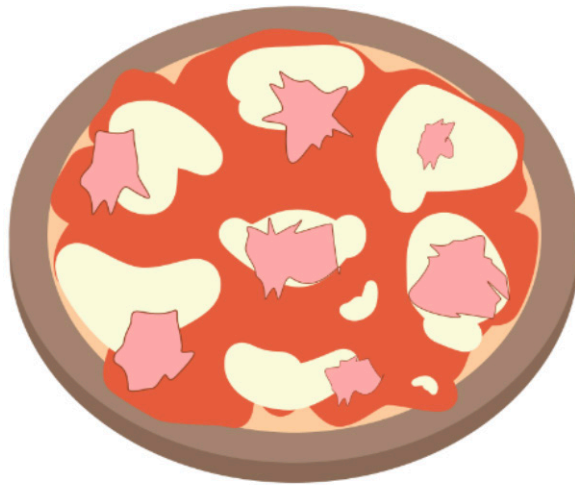
- Roll out dough.
- Spread tomato sauce on dough.
- Sprinkle cheese on tomato sauce.
- Sprinkle sliced peppers on top of cheese.
- Put pizza in oven at 350°C for 5 minutes.
- Take out of oven.



## Make a Pizza with a Procedure's Instruction

### 4. ANCHOVY PIZZA RECIPE

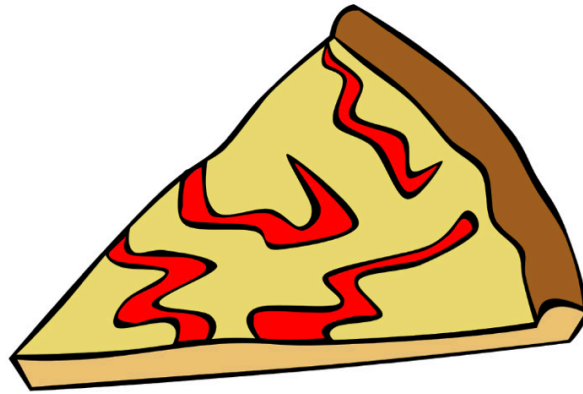
- Roll out dough.
- Spread tomato sauce on dough.
- Sprinkle cheese on tomato sauce.
- Sprinkle sliced anchovies on top of cheese.
- Put pizza in oven at 350°C for 5 minutes.
- Take out of oven.



## Make a Pizza with a Procedure's Instruction

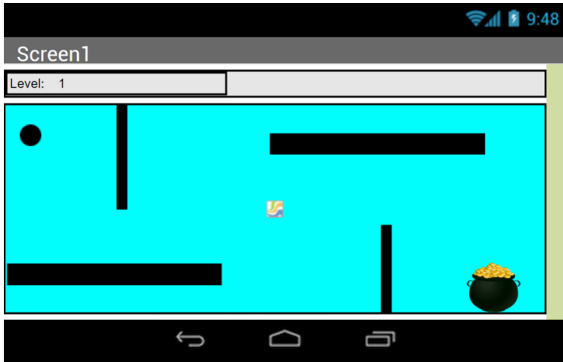
### 5. PIZZA RECIPE WITH OPTIONAL INGREDIENT

- Roll out dough.
- Spread tomato sauce on dough.
- Sprinkle cheese on tomato sauce.
- Sprinkle sliced optional ingredient on top of cheese.
- Put pizza in oven at 350°C for 5 minutes.
- Take out of oven.



## Lesson 4

## FIND THE GOLD APP

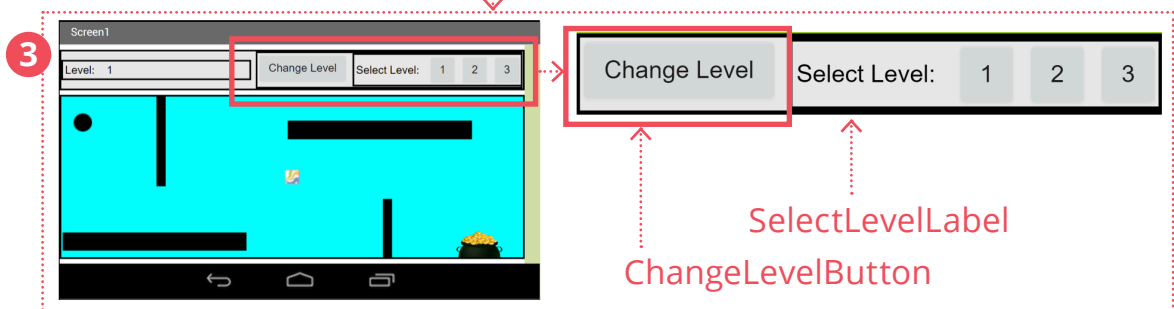
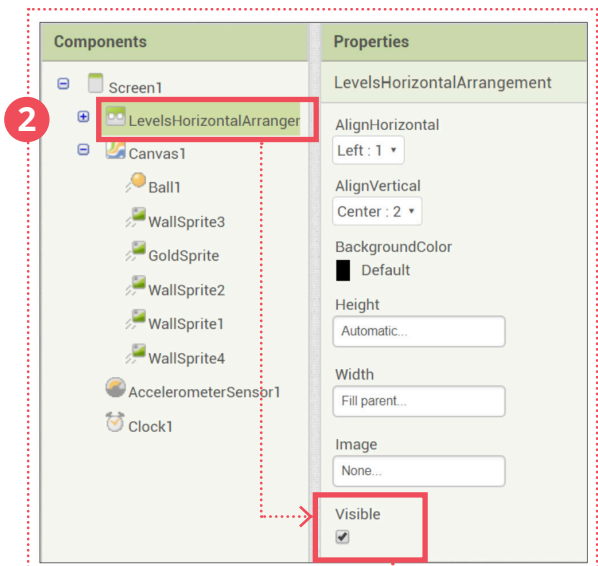


In this lesson, you will improve your game by adding buttons for selection of different levels.

 START HERE

In this lesson, you will make use of the **ChangeLevel** block by calling it in different places.

- ▶ **1** Open your Find The Gold project.
- ▶ **2** Under the **Components** tab, click the “+” next to **LevelsHorizontalArrangement** to reveal the list of components contained within.
- ▶ You will see it has two HorizontalArrangements contained within it – **DisplayLevelHorizontalArrangement** and **ChangeLevelHorizontalArrangement**.
- ▶ Click on **ChangeLevelHorizontalArrangement** and check the **Visible** box and see what happens in the Viewer.
- ▶ **3** Yes! There you go! Now you see the **ChangeLevelButton** and the select level components which have been added for you.



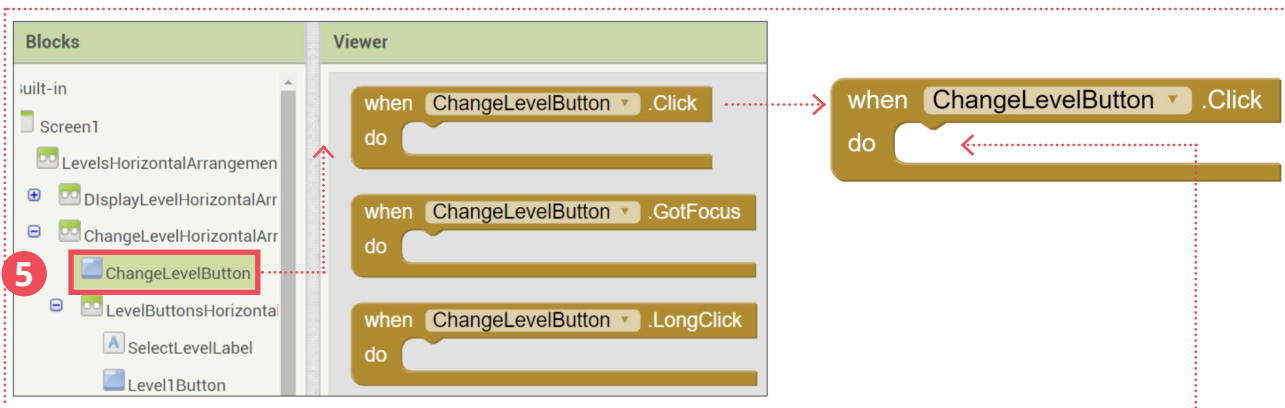
## CODE THE BUTTONS

By clicking the ChangeLevelButton, it will turn gray (become disabled) and then the level selection appears for users to click.

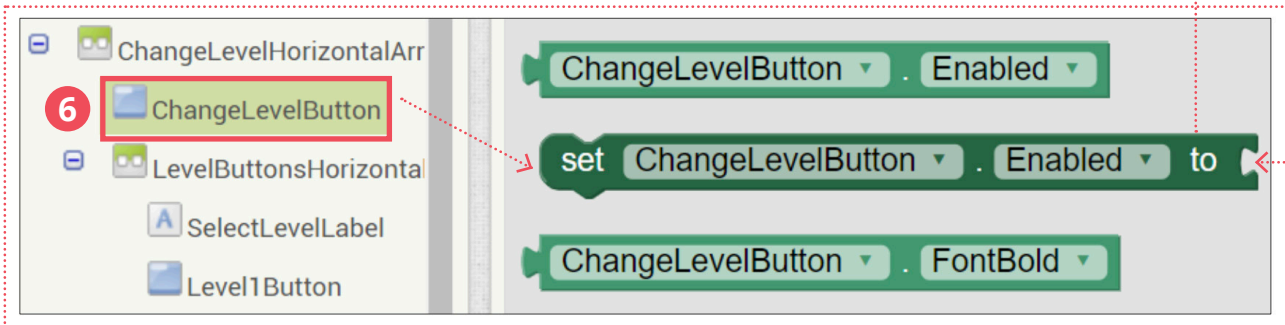
- ▶ **4** Click **Blocks** to go to the **Blocks Editor**.



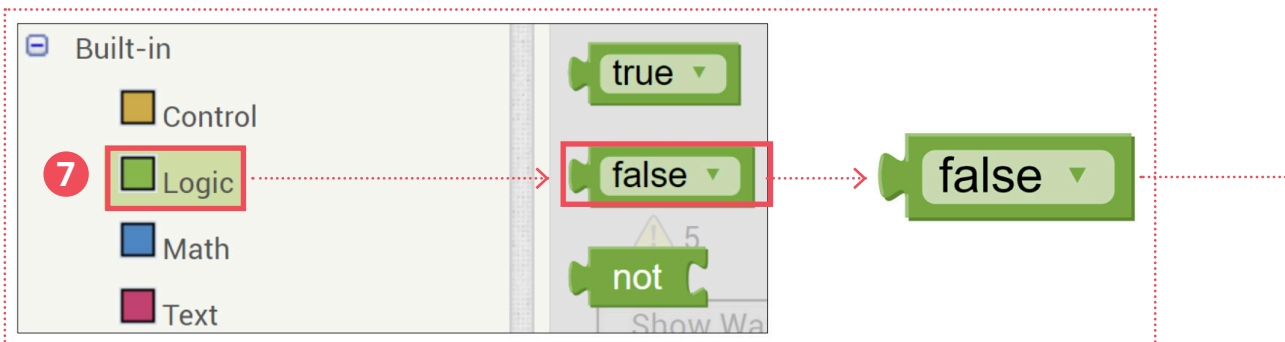
- ▶ **5** Drag a **when ChangeLevelButton.Click do** block from the **ChangeLevelButton** drawer.



- ▶ **6** Also drag a **set ChangeLevelButton.Enabled to** block and snap it into when **ChangeLevelButton.Click do** block.



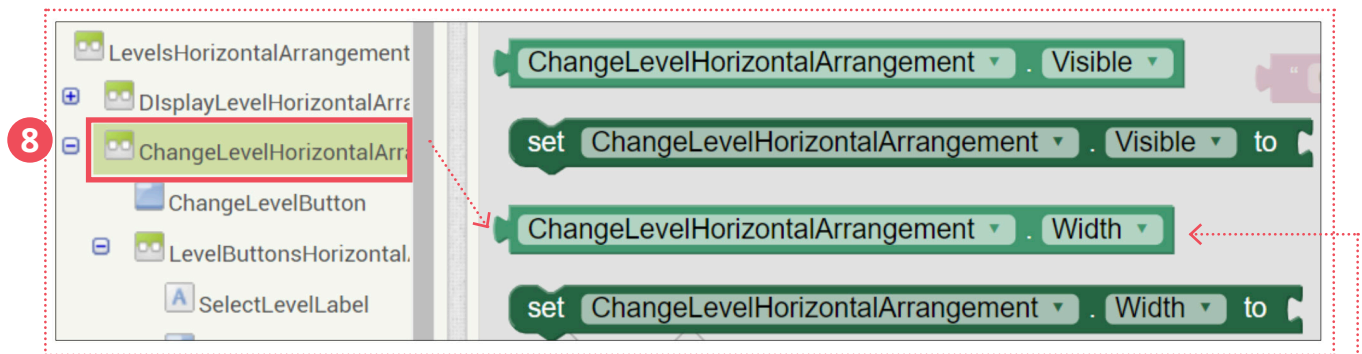
- ▶ **7** Drag a **false** block from the **Logic** menu and snap it into the set **ChangeLevelButton.Enabled to** block.



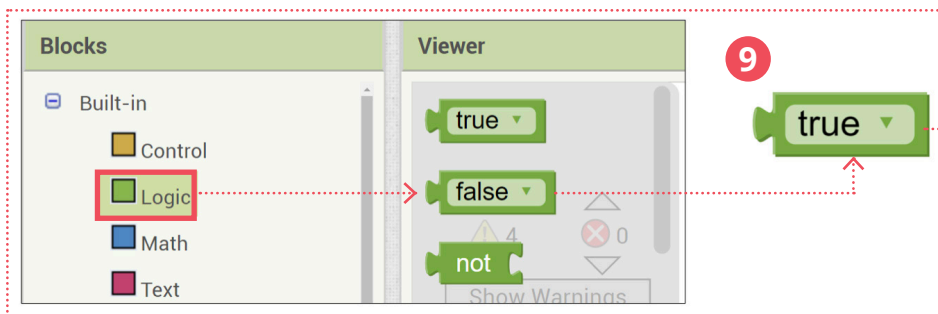
## LET'S CONTINUE

As the **"ChangeLevelButtonsHorizontalArrangement"** includes everything for the level selection, we should set it to visible so the user can choose a level.

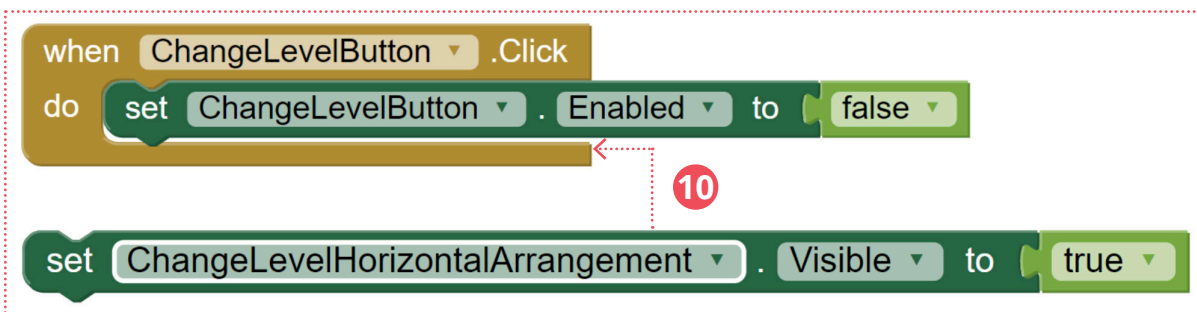
- ▶ **8** Drag a **set ChangeLevelButtonsHorizontalArrangement.Visible to** block



- ▶ **9** Drag a **true** block from the **Logic** menu and snap it into the **set ChangeLevelHorizontalArrangement.Visible to** block.



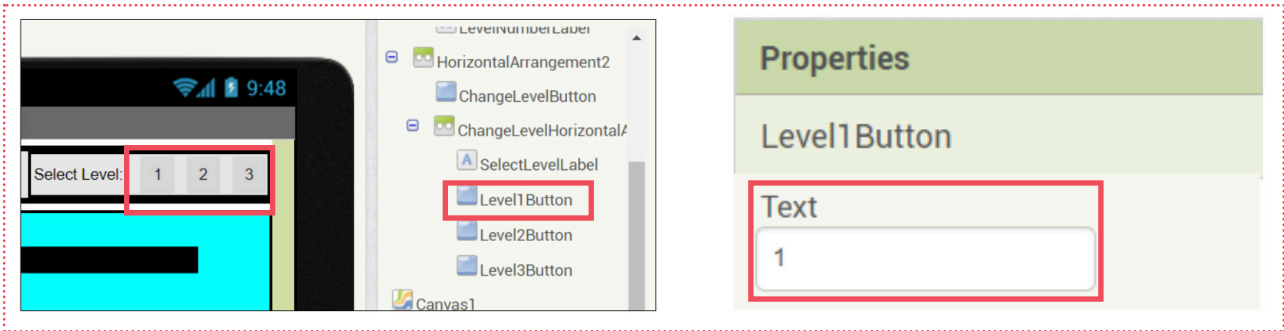
- ▶ **10** Snap the whole block into the **when ChangeLevelButton.Click do** block.



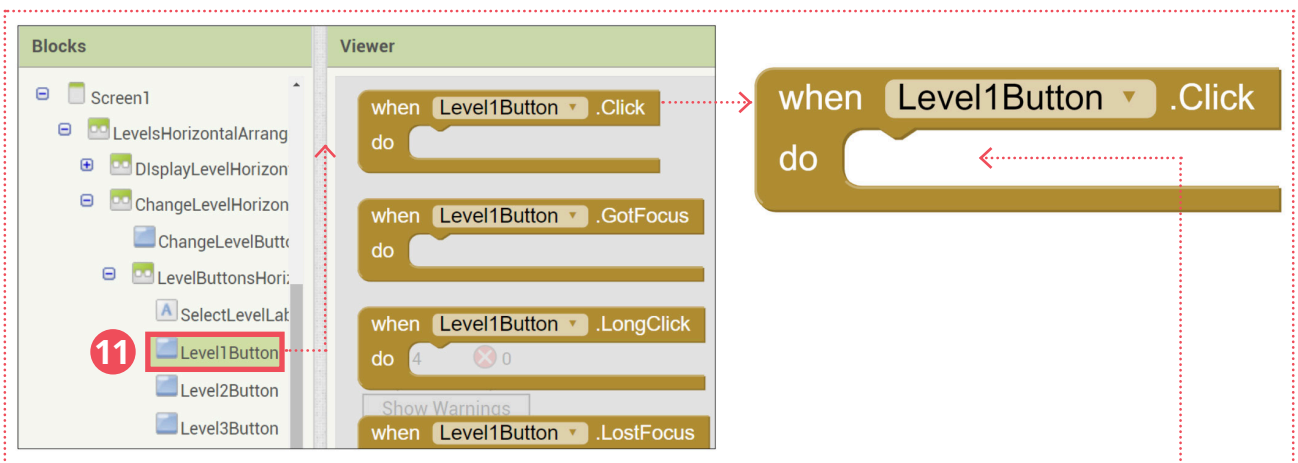
## CALL THE PROCEDURE

When the **Level 1 / 2 / 3 Buttons** appear, you can select and change the level directly by calling the **ChangeLevel** procedure.

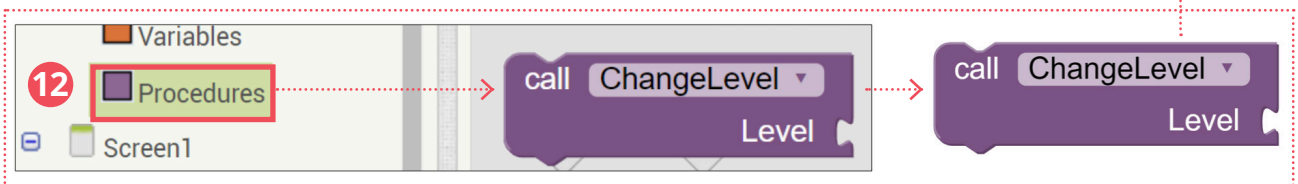
Also, the **Level1Button** Text property is the number **"1"** so we can simply use that text as the **"Level"** parameter in the procedure.



- ▶ **11** Drag a **when Level1Button.Click do** block and a **Level1Button.Text** block from the **Level1Button** menu.



- ▶ **12** Drag a **Call ChangeLevel** procedure with the **Level** parameter.



## CALL THE PROCEDURE (cont'd)

- ▶ **13** Drag a **Level1Button.Text** block and snap in as the **Level** parameter.

The screenshot shows a programming environment with a block palette on the left and a workspace on the right. In the palette, the 'Level1Button' block is highlighted with a red box and a red arrow points to it. In the workspace, a 'when Level1Button .Click' block is already present. A 'do' block containing a 'call ChangeLevel' block is being assembled. The 'Level' parameter of the 'call ChangeLevel' block is being connected to a 'Level1Button . Text' block. A second 'Level1Button . Text' block is shown below, with a red arrow pointing to its 'Text' field, which is being connected to the 'Level' parameter of the 'call ChangeLevel' block.

- ▶ **14** Now, can you code the buttons of Level 2 and Level 3? Hint: Duplicate and rename.

## UPDATE THE PROCEDURE

We just learnt how to disable the **ChangeLevelButton** and display the level selection when we click the **ChangeLevelButton**.

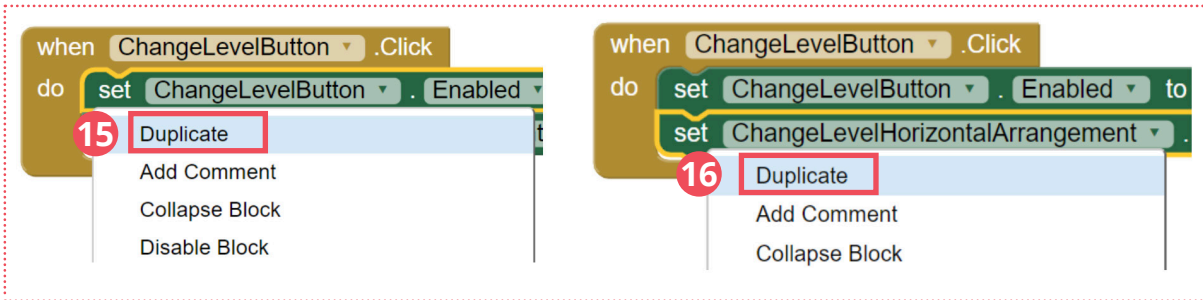
The screenshot shows a programming environment with a block palette on the left and a workspace on the right. In the palette, the 'ChangeLevelButton' block is highlighted with a red box and a red arrow points to it. In the workspace, a 'when ChangeLevelButton .Click' block is already present. A 'do' block containing two 'set' blocks is being assembled. The first 'set' block is 'set ChangeLevelButton . Enabled' to 'false'. The second 'set' block is 'set ChangeLevelHorizontalArrangement . Visible' to 'true'.



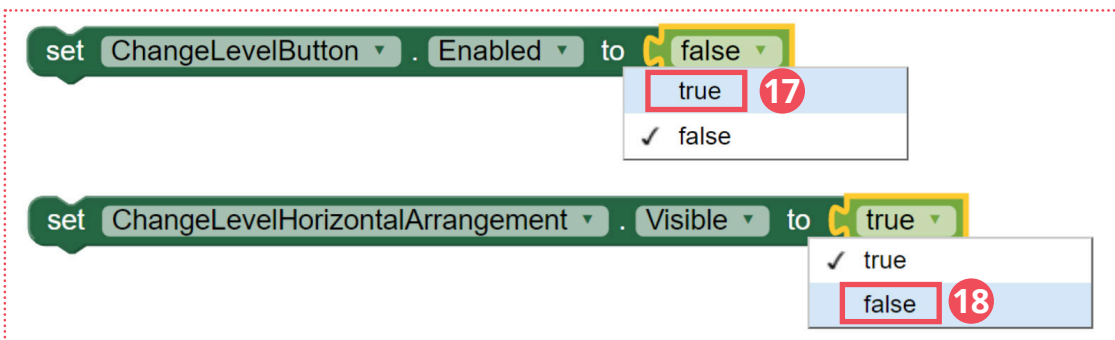
## LET'S CONTINUE

Every time we call the **ChangeLevel** procedure, we want to do the opposite. We want to enable the **ChangeLevelButton** but hide the level selection arrangement.

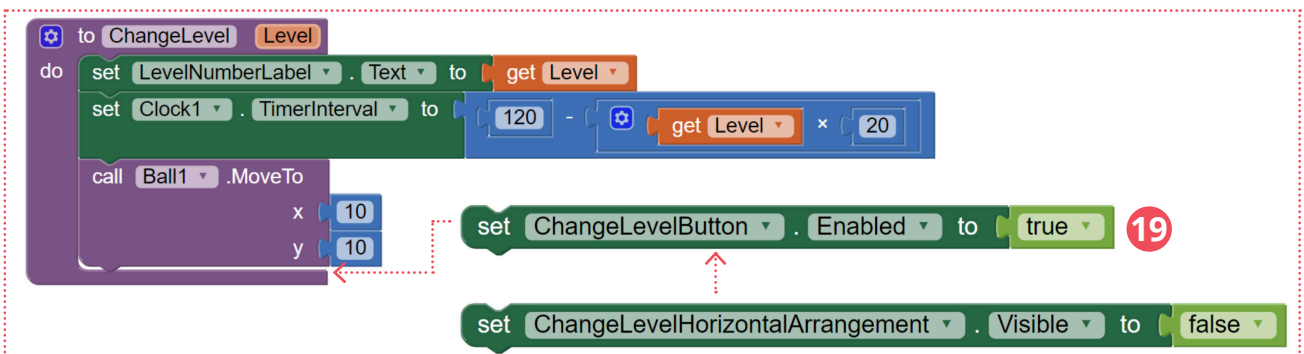
- ▶ **15 16** Since we'll do the opposite of the blocks we just created, let's **Duplicate** both of them.



- ▶ **17 18** Now change the logic to its reverse (false to true and true to false).



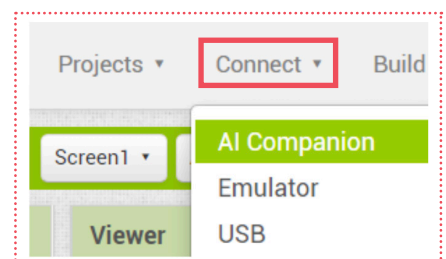
- ▶ **19** After completing those two blocks, snap them into the procedure.



- ▶ **20** Finally, Test and Debug using the MIT AI2 Companion app.

**CT Tips**

**Procedure:** Adding an input parameter to a procedure allows for more abstract procedures. Remember how we made pizzas with any ingredient? Similarly, we can change the app to any level with a parameter.



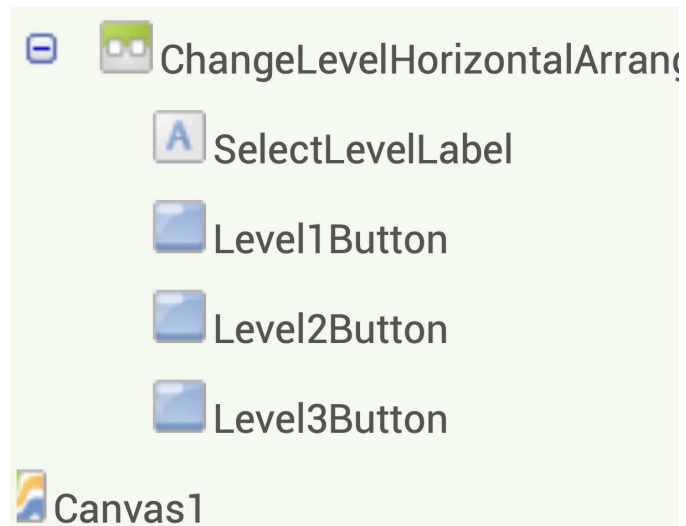
## COMPUTATIONAL THINKING CONCEPTS

The following are the computational thinking concepts learnt in Lesson 4.

### 1 Events



### 2 Naming



## COMPUTATIONAL THINKING PRACTICES

The following are the computational thinking practices used in this Unit.

### 1 Reusing and remixing:

- a) Reuse code/concepts from Level 1 Maze Game

### 2 Being incremental and iterative:

- a) Add Notifier component to the basic maze game app
- b) Update procedure to change the game level

### 3 Abstracting and modularizing:

- a) Add procedure to increase the game level
- b) Call the procedure with a parameter

### 4 Testing and debugging:

- a) Test that the ball moves correctly according to the tilt of mobile device
- b) Test that the ball goes back to the start position when it collides with obstacles (black walls)
- c) Test that a notifier is displayed when the ball collides with the Gold Pot
- d) Test that the correct actions are taken when user presses either "Play Again" or "Quit" buttons
- e) Test that the level changing works by user interaction and game progression.

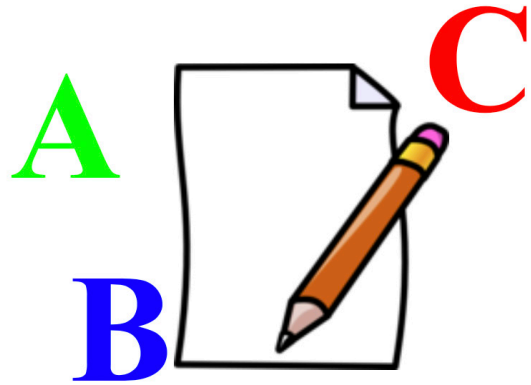
### 5 Algorithmic Thinking:

- a) Using the Clock component to trigger ball movement based on acceleration involves algorithmic thinking
- b) Make the ball move faster by adjusting the Clock.TimerInterval

## MIT APP INVENTOR FINAL PROJECT



In this project, you will make your own educational game app using what you have learnt so far! Take something you learnt making the Addition Game, Vocabulary Learning App or Find the Gold App and change it or add to it to make your own game!



### START HERE

**You and your partner will work together to build your app.**

- ▶ Using your design worksheet and “To-Do Checklist”, start making your app. Follow each step in your “To-Do Checklist”, and make sure each task is complete before moving on to the next step!
- ▶ Halfway through the project, your teacher will ask you to pair up with another group. Show each other your apps and fill out your feedback for your classmates using the Peer Feedback Worksheet.

#### Remember Rules for Pair Programming



1. Be respectful
2. Talk to one another about the work
3. Explain what you are doing
4. Think ahead and make suggestions
5. Switch roles often



1. Be a bossy navigator
2. Grab the driver’s mouse or keyboard

- ▶ Use your classmates’ feedback to improve your app.
- ▶ In the final class, you will present your apps to your classmates.

## COMPONENTS

### REMIXING CODE

You may use the **Backpack** to reuse or remix code.

- ▶ Find the code that you want to use from your existing app.
- ▶ Drag the block(s) to the Backpack.

```
when NumberImage1 .EdgeReached
  edge
do call NextQuestion
```



- ▶ Open your new project, click on the Backpack and drag out the blocks.

### PLAY SOUND

Your app can play any sound. Set the **Player1.Source** to the sound file. Then start the Player to play the sound.

```
set Player1 . Source to " yay.mp3 "
call Player1 .Start
```

### CLOCK TIMER

If you want code to run periodically in your app, use the **Clock** component.

- ▶ First, set the **TimerInterval** in the Clock Properties to set how often the timer event fires.
- ▶ Then code what should happen in the **Clock1.Timer** event.

```
when Clock1 .Timer
do
  set Ball1 . X to Ball1 . X + AccelerometerSensor1 . YAccel
  set Ball1 . Y to Ball1 . Y + AccelerometerSensor1 . XAccel
```

Properties	
Clock1	
TimerAlwaysFires	<input checked="" type="checkbox"/>
TimerEnabled	<input checked="" type="checkbox"/>
TimerInterval	100

## COMPONENTS

### ACCELEROMETERSENSOR

The AccelerometerSensor component detects shaking and acceleration change for the mobile device.

- ▶ The Shaking event indicates that the device is being shaken.

```
when AccelerometerSensor1 .Shaking
do
```

- ▶ The AccelerationChanged (xAccel, yAccel, zAccel) event indicates that the device is being tilted in the X, Y, and/or Z dimensions.

```
when AccelerometerSensor1 .AccelerationChanged
  xAccel yAccel zAccel
do
  set Ball1 . X to Ball1 . X - get xAccel
  set Ball1 . Y to Ball1 . Y + get yAccel
```

### NOTIFIER

Display a message to the user with the Notifier and respond based on which button they click.

- ▶ Use **call Notifier1.ShowChooseDialog** to display the message.

```
call Notifier1 .ShowChooseDialog
  message "You win!"
  title "Game Over"
  button1Text "Play Again"
  button2Text "Quit"
  cancelable false
```

In this example, the user wins so the 2 buttons give a choice to play again or quit.

- ▶ Then use **when Notifier1.AfterChoosing** to test which choice (which button) the user picked.

```
when Notifier1 .AfterChoosing
  choice
do
  if get choice = "Quit"
  then close application
  else set Clock1 . TimerEnabled to true
```

If the user chooses "Quit", close the application.

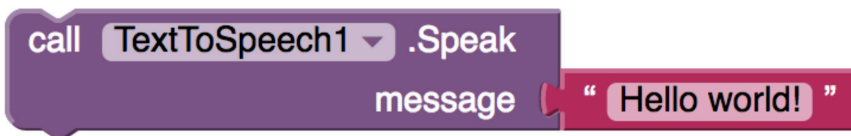
Otherwise, in this case, turn the Clock on.

## COMPONENTS

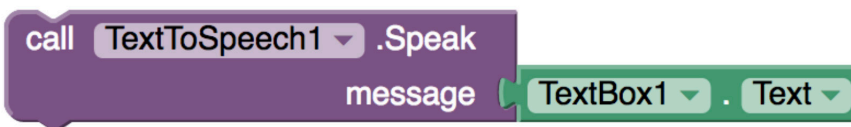
### TEXT TO SPEECH

Your app can speak any text string you want with the **TextToSpeech** component.

- ▶ You can specify a text block with text you want the app to say.



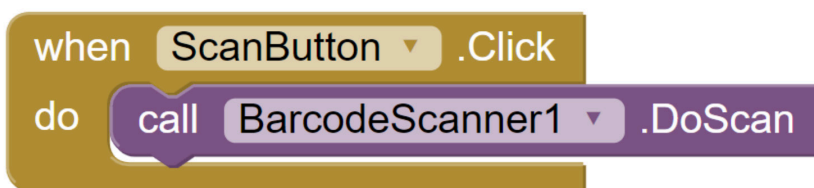
- ▶ Your app can also speak the contents of a textbox, or any other user interface component!



### BARCODE SCANNER

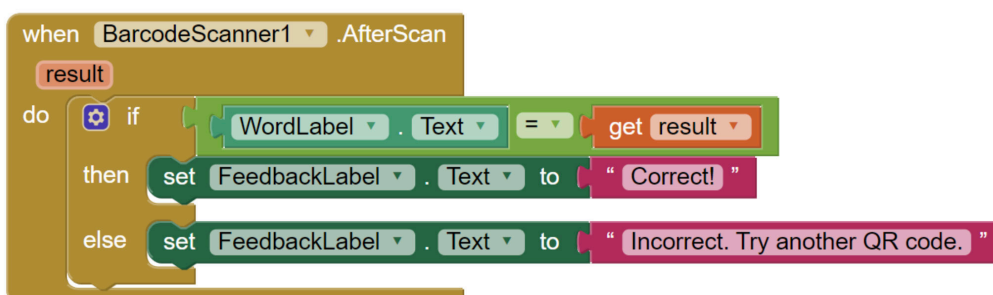
If you want to scan a barcode in your app, use the **BarcodeScanner** component.

- ▶ First, call **BarcodeScanner.DoScan**.



This example starts the barcode scanner when the user clicks the button.

- ▶ Then when **BarcodeScanner1.AfterScan** event gets the scanned result and executes the blocks contained within.



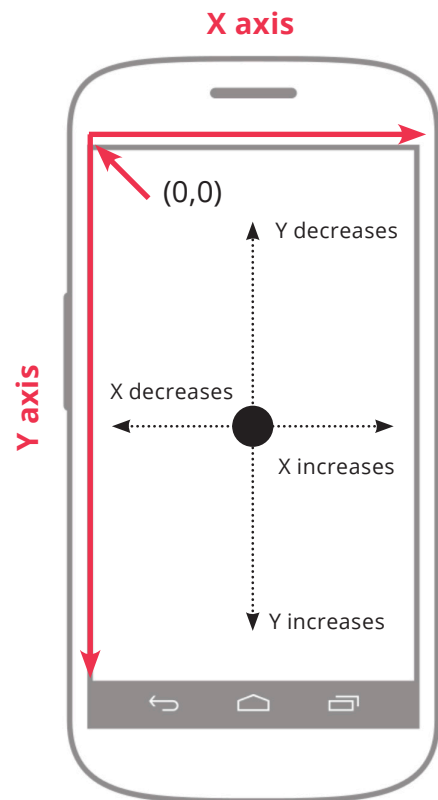
## COMPONENTS

### CANVAS AND IMAGESPRITES

You must add a **Canvas** component to your app, and an **ImageSprite** or **Ball** components to the **Canvas**.

You can set the position of **ImageSprites** and **Balls** by their **X,Y** coordinates, just like in the Cartesian coordinate system. One difference is that the origin (0,0) is at the top left corner of the screen. X increases as it moves to the right of the screen.

Y increases as it moves down the screen.



### ImageSprite Properties

**Heading** is the direction (0 - 360).

**Height** and **Width** can be set to resize your sprite.

**Interval** is how often the Sprite moves (in milliseconds)

**Picture** can be set to an image file uploaded to your project.

**Speed** is how many pixels the sprite moves each interval.

**X** and **Y** are the positions of the ImageSprite.

Properties	
HorizontalWall1	
Enabled	<input checked="" type="checkbox"/>
Heading	0
Height	20 pixels...
Width	200 pixels...
Interval	100
Picture	horizontalwall.jpg...
Rotates	<input checked="" type="checkbox"/>
Speed	0.0
Visible	<input checked="" type="checkbox"/>
X	-5
Y	56
Z	1.0

**PaintColor** lets you change the Ball's color.

**Radius** determines the size of Ball.

### Ball Properties

Properties	
RedBall	
Enabled	<input checked="" type="checkbox"/>
Heading	0
Interval	100
PaintColor	<span style="color: red;">■</span> Red
Radius	2

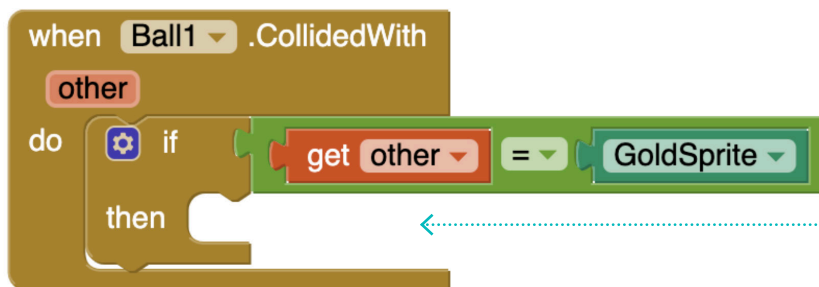


## COMPONENTS

### IMAGESPRITE COLLISION

To detect if two ImageSprites are touching, use the **when ImageSprite.CollidedWith** block.

- ▶ This block tests if a Ball and an ImageSprite collide.

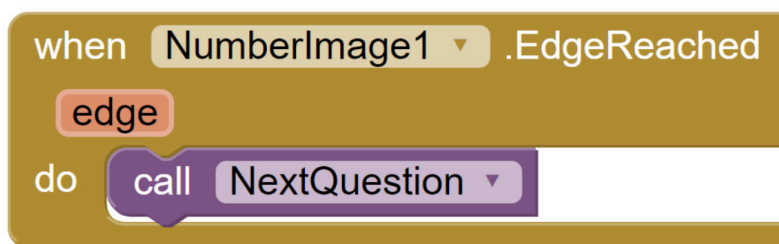


Ball collides with GoldSprite.

- ▶ Fill in what should happen when the Ball or ImageSprites collide in the **then** section.

### IMAGESPRITE REACHES EDGE

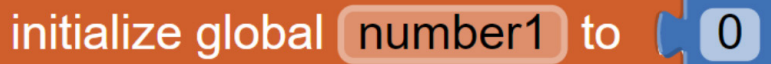
If you want to perform an action when an ImageSprite reaches the edge of the screen, use the **when ImageSprite.EdgeReached** block.



## CONCEPTS

### VARIABLES

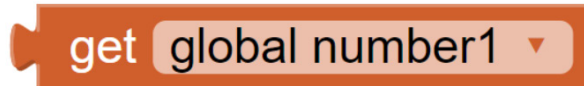
Your app can use variables to store any information in your app. **initialize global \_\_ to** is used to initialize and name the variable.

A Scratch 'initialize global' block with the variable name 'number1' and the value '0'.

▶ **set global \_\_** is used to change the value of the variable.

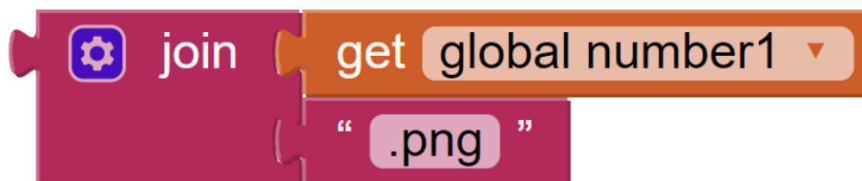
A Scratch 'set global' block with the variable name 'number1' and a 'random integer from 1 to 9' block as the value.

▶ **get global \_\_** is used to get the current value of the variable.

A Scratch 'get global' block with the variable name 'number1'.

### JOIN

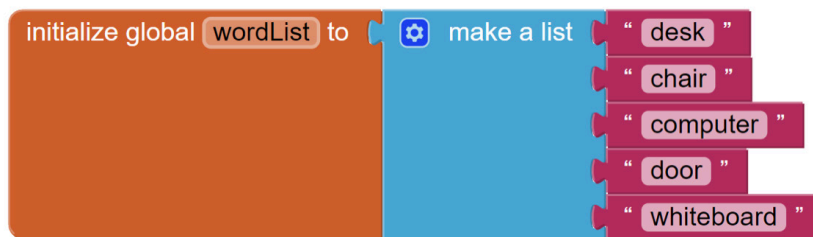
If you want to join multiple strings in your app, use the **join** block.

A Scratch 'join' block with a 'get global number1' block and the string '.png' as inputs.

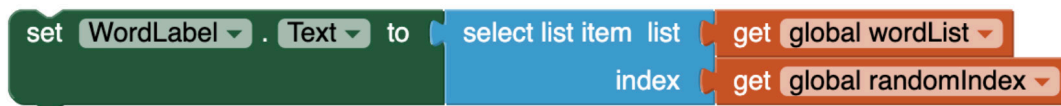
## CONCEPTS

### LISTS

Lists allow you to store multiple values in a single variable.



- ▶ **Select list item** is used to extract an item from a list, using its index.



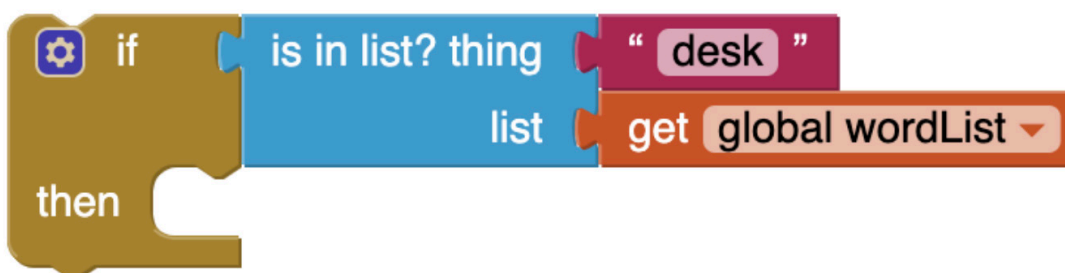
- ▶ **length of list** returns the length of the list, or how many elements are in a list.



- ▶ **Is list empty?** is true if the list is empty, and false if it contains elements.



- ▶ **Is thing in list?** is true if thing is an element in the list, and false if it is not.



## GAME FEATURES

### KEEPING SCORE

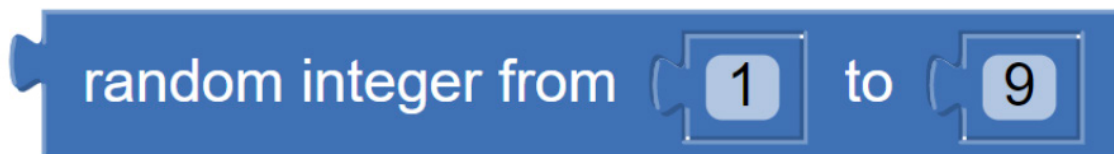
If you want to provide scoring in your app, you can add a Label called “Score”, and then use the **set Score.Text to** block.

- ▶ Use **Score.Text + 1** to add to the score, **Score.Text - 1** to subtract from the score.



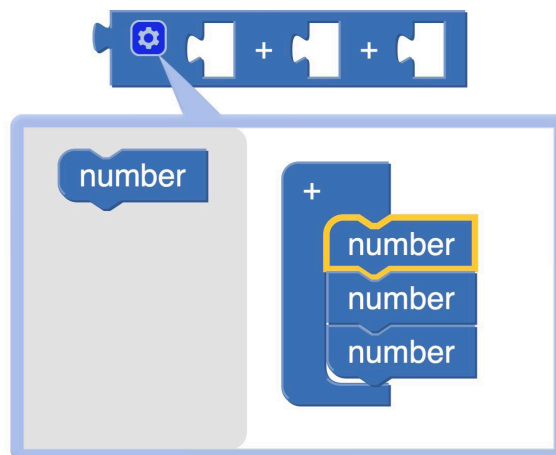
### GENERATE RANDOM NUMBERS

If you want to generate a random number, you use the **random integer from** block.



### CALCULATE THE SUM OF NUMBERS

Use the addition (+) block to sum several numbers.



# Educational Game App Design Worksheet

Name: \_\_\_\_\_

Class: \_\_\_\_\_ Date: \_\_\_\_\_

**In this unit, you will design your app to help users learn or practice a new skill.**

## App Requirements:

New Components (pick at least two)	Concepts (pick at least two)	Required
<ul style="list-style-type: none"><li>• <b>Player</b></li><li>• <b>Button</b></li><li>• <b>TextBox</b></li><li>• <b>BarcodeScanner</b></li><li>• <b>Accelerometer</b></li><li>• <b>TextToSpeech</b></li><li>• <b>ImageSprite</b></li><li>• <b>Notifier</b></li></ul>	<ul style="list-style-type: none"><li>• <b>Lists</b></li><li>• <b>Variables</b></li><li>• <b>Repetition</b></li><li>• <b>Procedures</b></li></ul>	<ul style="list-style-type: none"><li>• <b>Conditionals</b></li></ul>

## 1. Description of Your Game App

Describe the game app you want to create.

---

---

---

---

---

---

---

---

# Educational Game App Design Worksheet

## 2. Screen Design

Screenshots - draw pictures of what your app's screen(s) will look like.



# Educational Game App Design Worksheet

## 3. Feature List

Check the features you will include with your app, by marking with a ✓:

<b>Components (Remember to pick at least two!)</b>		<b>Concepts (Remember to pick at least two!)</b>	
Player		Lists	
Button		Variables	
TextBox		Repetition	
BarcodeScanner		Procedures	
Accelerometer			
TextToSpeech			
ImageSprite			
Notifier			

# Educational Game App Design Worksheet

## 4. Component List

List all the components you will need for your app.

### Screen1

Component Type (Button,TextBox, etc)	Name	Special Properties (font, colour, alignment, etc)

### Describe how your components work together.

E.g. When Button1 is clicked, do ...

---

---

---

---

---



**Copyright © The Hong Kong Jockey Club Charities Trust 2022**

All rights reserved. The intellectual property subsisting in the material published is owned by The Hong Kong Jockey Club Charities Trust. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher.

Publisher: The Hong Kong Jockey Club Charities Trust

July 2022 Edition

ISBN: 978-988-74553-5-6

Address: 1 Sports Road, Happy Valley, Hong Kong

Website: <https://www.coolthink.hk>

Email: [coolthink@hkjc.org.hk](mailto:coolthink@hkjc.org.hk)

Created and Funded by



The Hong Kong Jockey Club Charities Trust

Content Co-created by



Design and Produced by



**Disclaimer**

All content in this book is intended for educational use and non-profit marking purpose only. The publisher shall not be held liable for any claims or losses arising from any use of the content. Websites and hyperlinks in the references are published or operated by independent organizations including but not limited to MIT Scratch, MIT App Inventor and YouTube. We make no warranties whatsoever and we shall not be deemed to have endorsed, recommended or approved any information, products or services contained in or offered through such sites.

**{oo/Think** @JC >  
賽馬會運算思維教育  
Inspiring digital creativity 啟發數碼創意

ISBN 978-9-88-745535-6



9 789887 455356 >