Course Description

This course introduces a wide range of smart applications in various situations. It focuses on the use and application of data. Students will utilize and strengthen their logical and mathematical thinking skills to design smart application solution based on data collected from daily life. Students will also develop their growth mindset and transferable skills during their exploration and learning journey in this course. Students will explore and elaborate on various physical components and consider how to program them, create digital artefacts, and demonstrate the prospect of technologies and how it drives to a smart society.

This course addresses the following key themes:

- Data in everyday life
- Data and smart applications
- Data and creative culture

Content

Lesson	Area of Enquiry
Lesson 1 – Noise Detector	Smart Automation
Lesson 2 – Voice Reactive Lights	
Lesson 3 – Remote Classroom Polling	
Lesson 4 – Houseplant Care	
Lesson 5 – Smart Notifier	
Lesson 6 – Parcel Locker	
Lesson 7 – Intelligent Vehicles	
Lesson 8 – Bus Tracker	
Lesson 9 – Colors Hunter	Game Design
Lesson 10 – Opposite Game	
Lesson 11 – Catch Fish Carnival Game I	
Lesson 12 – Catch Fish Carnival Game II	
Lesson 13 – Dodge the Bird I	
Lesson 14 – Dodge the Bird II	

Target Audience

Age groups: 11~14 years old (11~12 years old)

Level of Difficulty

Introductory, prior block-based programming knowledge is necessary

Effort

40~45 minutes per lesson, 14 lessons in total for one semester/term

Lesson 1

Noise Detector

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Identify the basic characteristics and types of data.
- Clarify the importance and common use of data in everyday life.
- Use sensors to retrieve data, and display data on the screen.
- Write programs to create a noise detector.

Key Focus

- Basic understanding of the characteristics and types of data, and the importance of data in everyday life.
- Using sensors to retrieve data and displaying data on the screen.
- Identifying different types of data and their functions.

Content Standards

Type	Indicator	Standard
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
CSTA	2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
ISTE	5b	Students collect data or identify relevant data sets, use digital tools to analyze them, and represent data in variousways to facilitate problem-solving and decision-making.

Preparation

For the Teacher:

- A laptop or desktop with mBlock installed
- A CyberPi kit
- Pens and paper

For Students:

- Proficiency in block-based coding
- Basic knowledge of sensors

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Have students do an in-class survey and collect data, preparing them for the later introduction to data.
5 minutes	**Section 2 – Explore** • Introduce a scenario where a noise detector could be used, and have students think about noise pollution in their life.
5 minutes	**Section 3 – Explain** • Have students brainstorm possible solutions to deal with the data they've collected, and design features they want.
20 minutes	**Section 4 – Elaborate** • Give students the freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Have students think of any situation in which they will use their project.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Have students do an in-class survey and collect data, preparing them for the later session about what data is.

Procedures:

- Hand out pens and paper to students. Have students work in pairs. Ask them to investigate the topic by asking classmates on the following questions:

  (1) Where would you like to spend your weekends most?

  (2) How many times do you go to the library per month?

  (3) Have you ever seen any uncivil behaviors?

- Give students 1 minute to work on their survey.
- Pick 2 or 3 students to share their survey.

- **Ask:** What's your first thought when you hear "data"? What is data?
- Explain:
  - **Types of data:** The answers to the three questions above could all be considered data. Data is distinct information that is formatted in a special way. Birthday, hobby, height, weight and others. All these are data. Data exists in a variety of forms, like images, sounds, texts, symbols and more.
  - **Use of data:** Data could be used to convey messages, e.g. queue management system, weather report, bus arrival notification...
  - **Summary:** Data helps us solve problems in many ways.

Section 2 – Explore
[5 minutes]

Objective:

- Introduce a scenario where a noise detector could be used, and have students think about noise pollution in their life.

Procedures:

- Discuss a scenario where noises are annoying.
- **Ask:** If you're on a bus and someone is talking aloud, how will you feel?
- **Ask:** Suppose we now have a device that's monitoring sounds around. To reduce some noises like this, how should we design the device? What features must the device have? You have 1 minute to discuss with your partner.
- Pick 2 students to share their thoughts.
- Categorize the features based on students' discussion

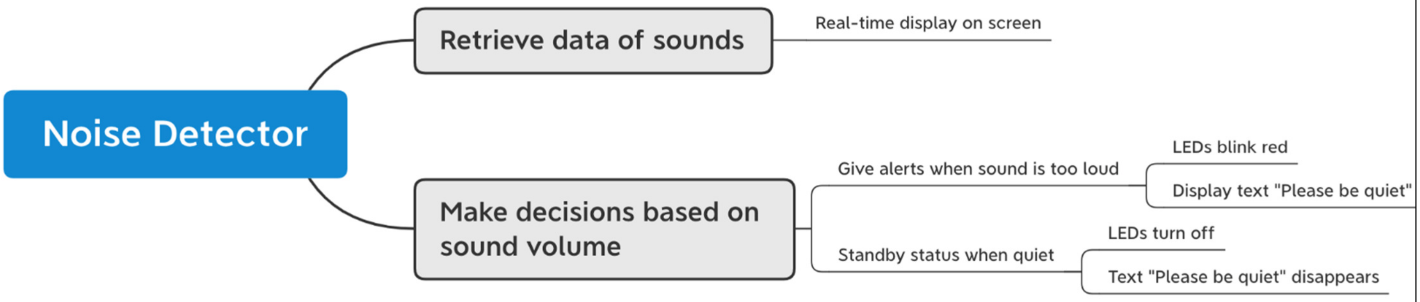
Section 3 – Explain
[5 minutes]

Objective:

- Have students brainstorm possible solutions to deal with the data they've collected, and design features they want.

Procedures:

- Show students how the noise detector works.
- Work with students to analyze what features a noise detector must have, and draw a mind map.


Noise Detector
Retrieve data of sounds
Real-time display on screen
Make decisions based on sound volume
Give alerts when sound is too loud
LEDs blink red
Display text "Please be quiet"
Standby status when quiet
LEDs turn off
Text "Please be quiet" disappears

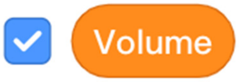
Section 4 – Elaborate
[20 minutes]

Objective:

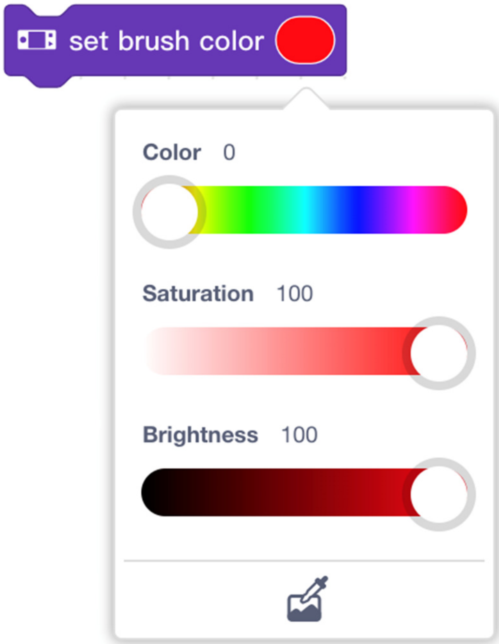
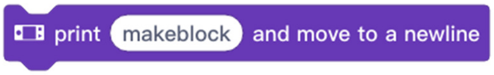
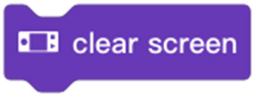
- Give students the freedom to explore further so that they can enhance their programming skills.

Procedures:

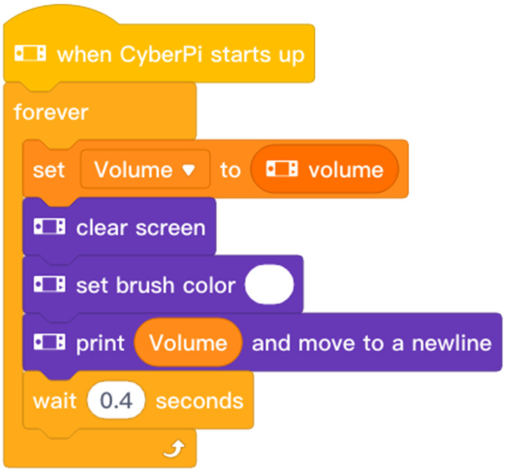
- Show students where these modules are on CyberPi: LED Screen, Buttons, Joystick, Switch, Sensors, LED Indicators.
- Teach students how to start their CyberPi and connect the device to mBlock.
- Add the device CyberPi in mBlock, and start programming it.
- Display sound volume
  - Use CyberPi's sound sensor to detect sound volume. Tick the coding block below to display volume on the stage in mBlock.



  - To display sound volume on CyberPi's screen, you need to use the following three blocks

Blocks palette	Coding blocks	Feature
Display	 set brush color Color 0 Saturation 100 Brightness 100	Use this block and drag the sliders to decide the color of your content displayed on the screen.
	 print makeblock and move to a newline	Use this block to decide what content you're going to display on the screen.
	 clear screen	Use this block to remove all the content on the screen.

○ Program CyberPi to display sound volume:



```
when CyberPi starts up
forever
    set Volume ▾ to volume
    clear screen
    set brush color
    print Volume and move to a newline
    wait 0.4 seconds
```

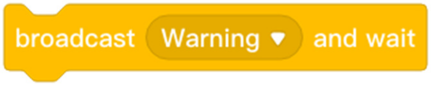
○ *NOTE:*

    ① *Remove all the existing content on the screen at the very beginning to make sure the new content is clearly displayed.*

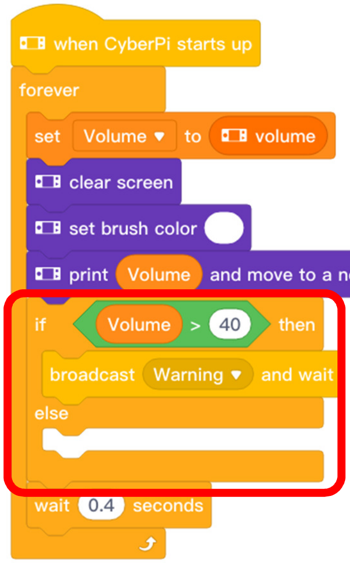
    ② *Have students use the device to directly retrieve volume and observe what happens.*

  - **Explain:** The device has an in-built sensor to detect sound volume in real time, so the volume value retrieved will keep fluctuating all the time. As a result, you may find it much harder to see an exact volume value on the screen. To slow down the changes of value, you can set up a variable named "volume" to store the volume value retrieved.

- Make decisions based on the sound volume
  - Decide whether to give alerts or not.
  - Use the block **broadcast () and wait** to send alerts.

Blocks palette	Coding blocks	Feature
● Events	 broadcast Warning ▾ and wait	Use this block to broadcast messages and wait until the scripts activated by the broadcast stop running before beginning to execute the next command.

  - Relying on the mind map, program CyberPi to make decisions based on volume:



```
when CyberPi starts up
forever
    set Volume to volume
    clear screen
    set brush color
    print Volume and move to a newline
    if Volume > 40 then
        broadcast Warning and wait
    else
    wait 0.4 seconds
```

- *NOTE:*

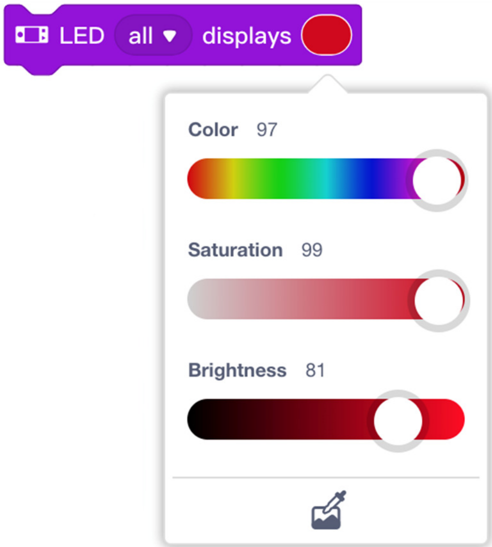
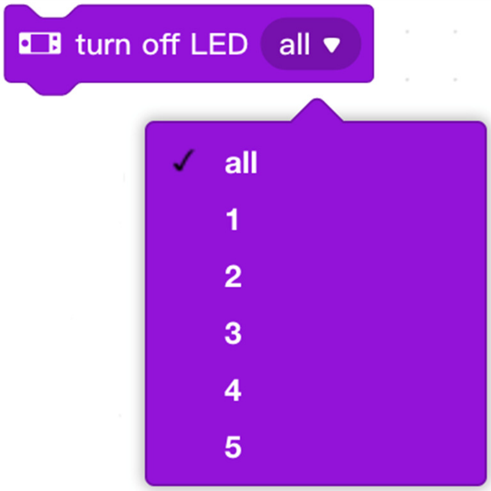
  The example program above uses "40" as a threshold. But students can change the threshold to another value as they'd like to.
- Define the alerts and standby
- Based on what we observed, CyberPi will give the following alerts when the volume is too high.

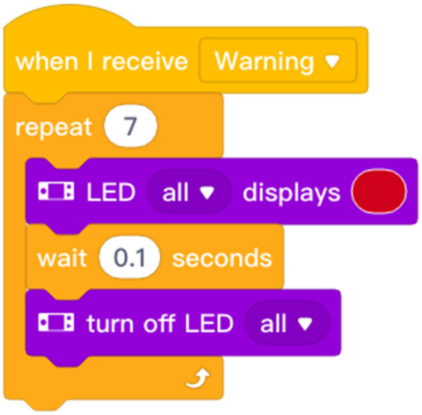
  ① LEDs blink red;

  ② The screen displays the text "Please be quiet".

- Program CyberPi to give alerts:
  - LEDs blink red:

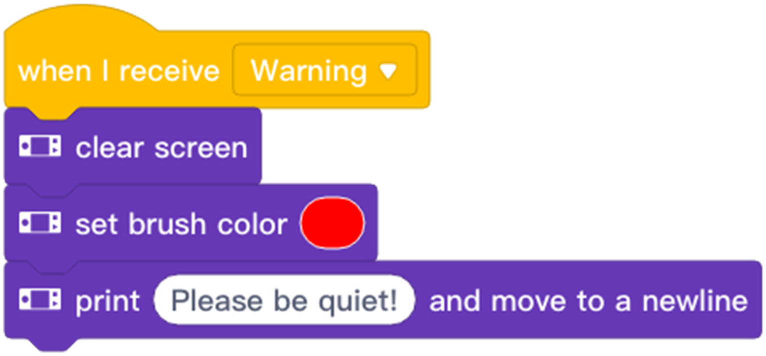
Use the blocks below:

Blocks palette	Coding blocks	Feature
LED	 LED all ▾ displays Color 97 Saturation 99 Brightness 81	Use this block to decide what color the LED turns on and which LED to turn on.
	 turn off LED all ▾ ✓ all 1 2 3 4 5	Use this block to turn off all LEDs or a specific LED.



```
when I receive Warning ▾
repeat 7
    LED all ▾ displays
    wait 0.1 seconds
    turn off LED all ▾
```

- Program CyberPi to display the warning text "Please be quiet" on its screen:



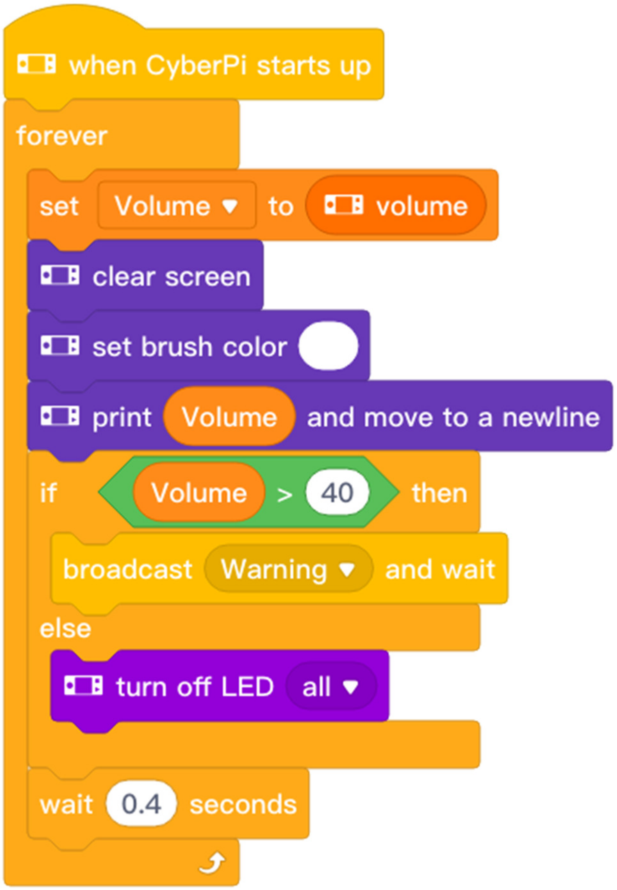
```
when I receive [Warning ▾]
clear screen
set brush color ( )
print (Please be quiet!) and move to a newline
```

- Standby status:

  ① LEDs turn off;

  ② Warning text disappears.

  - Program CyberPi to define its standby status:



```
when CyberPi starts up
forever
    set [Volume ▾] to (volume)
    clear screen
    set brush color ( )
    print (Volume) and move to a newline
    if <(Volume) > (40)> then
        broadcast [Warning ▾] and wait
    else
        turn off LED [all ▾]
    wait (0.4) seconds
```

  - *NOTE:*

    The program already includes a coding block for clearing the screen, so we don't need to clear the screen again here.

Section 5 – Evaluate
[5 minutes]

Objective:

- Have students think of any situation in which they will use their project. Make them rethink about incivility in public space and the application of data in everyday life.

Procedures:

- Pick 1 or 2 students to showcase their noise detector.
- Where do we need to use noise detectors? Where and when should we be careful about our volume? Ask students to have a discussion.
- **Ask:** Can you think of any other applications of data in daily life? What data are you familiar with in your everyday life?

Lesson 2

Voice Reactive Lights

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Display a bar chart on the screen
- Change lighting effects and bar chart based on the sound levels
- Use mind maps to analyze how to complete a project
- Write programs to complete the Voice Reactive Lights project

Key Focus

- Displaying a bar chart on the screen
- Changing lighting effects based on sound levels

Content Standards

Type	Indicator	Standard
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
CSTA	2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
ISTE	5b	Students collect data or identify relevant data sets, use digital tools to analyze them, and represent data in various ways to facilitate problem-solving and decision-making.

Preparation

For the Teacher:

- A laptop or desktop with mBlock 5 installed
- A CyberPi kit

For Students:

- Prior knowledge of CyberPi’s major features and how to collect data with CyberPi’s sensors

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Show pictures and play videos to demonstrate how to apply lights and sounds in a show.
5 minutes	**Section 2 – Explore** • Bring up the question and have students brainstorm.
5 minutes	**Section 3 – Explain** • Focus on the interaction between sound, light, and fountain, and prepare students for the Elaborate part.
20 minutes	**Section 4 – Elaborate** • Give students the freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Play some music and invite students to present their projects.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Show pictures and play videos to demonstrate how to apply lights and sounds in a show.

Procedures:

- Have students search for and compare traditional fountains and musical fountains.
- Have students answer the questions: Which type of fountains do you like? Why?
- Introduce the working principles and characteristics of musical fountains.

  - Music, lights, and fountain are programmed to react to each other.

  - The lights and fountain are programmed to react to the music

Section 2 – Explore
[5 minutes]

Objective:

- Bring up the question and have students brainstorm.

Procedures:

- Discuss musical fountains.
- Have students answer the question: How do the music, fountain, and lighting fit together?
- Conclude the following:

  Louder music means ① lights in a darker color, ② higher spout, and ③ faster changes in shapes.

  Softer music means ① lights in a lighter color, ② lower spout, and ③ slower changes in shape.

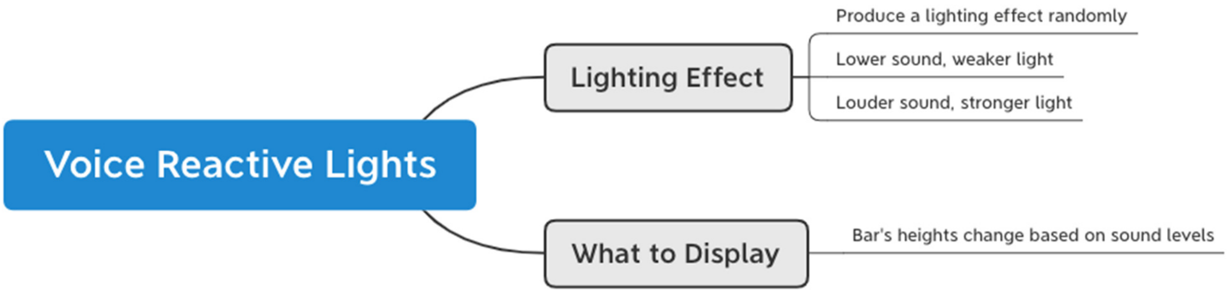
Section 3 – Explain
[5 minutes]

Objective:

- Focus on the interaction between sound, light, and fountain, and prepare students for the Elaborate section.

Procedures:

- Tell students: Though we can't make musical fountains, we can use CyberPi to make Voice Reactive Lights.
- Analyze with the students what features the project has and draw a mind map.


Voice Reactive Lights
Lighting Effect
Produce a lighting effect randomly
Lower sound, weaker light
Louder sound, stronger light
What to Display
Bar's heights change based on sound levels

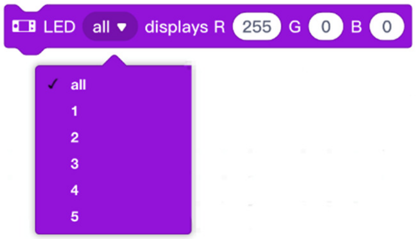
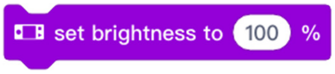
Section 4 – Elaborate
[20 minutes]

Objective:

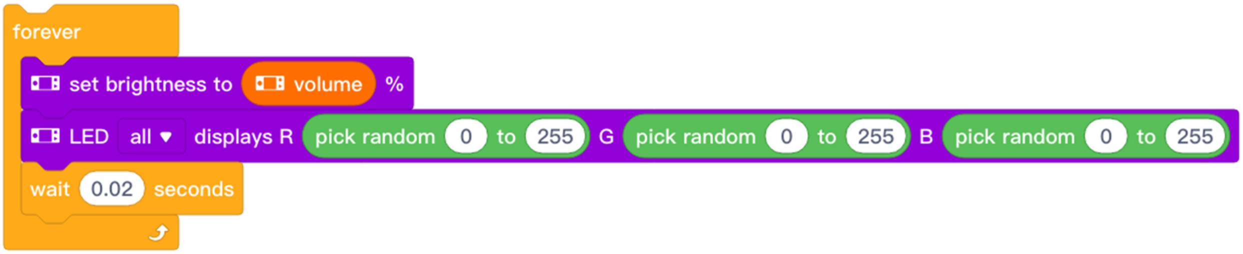
- Give students the freedom to explore further so that they can enhance their programming skills.

Procedures:

- Remind students that they need to connect the device to the computer and select **Upload** mode.
- Recap what hardware components CyberPi has and demonstrate how to detect sound level with the sound sensor.
- Design lighting effects:
    - **Tell students:** To change the lighting effects, we need these coding blocks.

Category	Coding block	Feature
LED	 LED all ▾ displays R 255 G 0 B 0 ✓ all 1 2 3 4 5	This lighting block sets the color of a specific LED with specified RGB values. Each color component ranges from 0 to 255. The greater values produce brighter lights.
	 set brightness to 100 %	This block defines the brightness of the LEDs. The value ranges from 0 to 100. Increase the value to make the light stronger and reduce the value to make the light weaker.

- **Say:** We've learned that we could use the volume to decide the lighting and other effects. So, why don't we try using the sound sensor to detect the sound level?



```
forever
    set brightness to (volume) %
    LED all ▾ displays R (pick random 0 to 255) G (pick random 0 to 255) B (pick random 0 to 255)
    wait 0.02 seconds
```

- *NOTE:*

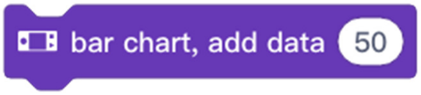
  *① To have lighting effects randomly produced, use **pick random ( ) to ( )** to set the RGB values.*

  *② To ensure a seamless transition between different light effects, use **wait ( ) seconds**.*

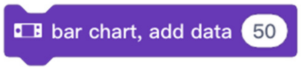
  (Instruct students to include the wait block in the program, then remove it, and compare the results.)

③ *The lighting starts changing as soon as the program runs.*

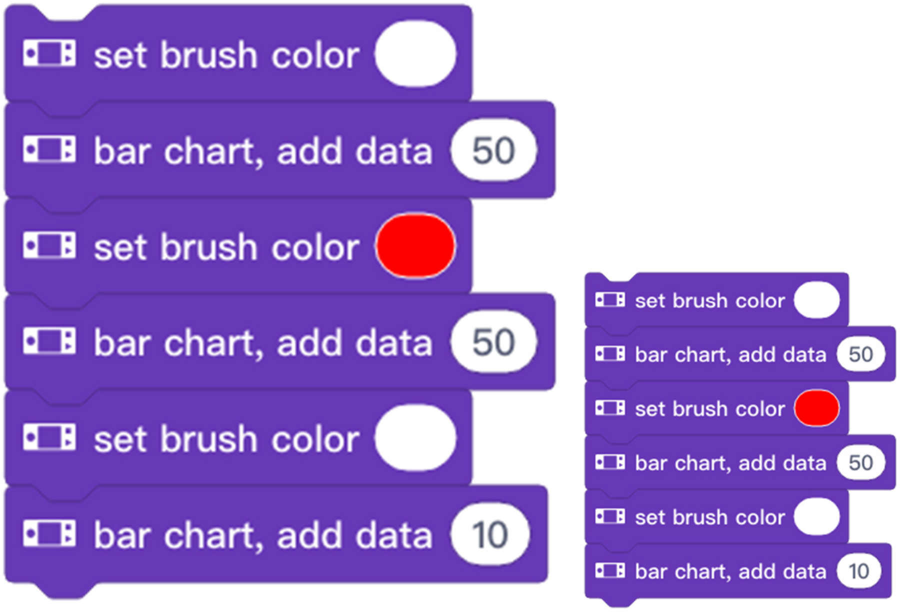
- Design what to display on the screen.
  - Use chart-related blocks to show the sound levels on the display.

Category	Coding block	Feature
Display	 bar chart, add data 50	This block draws a bar. It defines the number that a bar represents and shows the bar on the screen.

  - *NOTE:*

① *To draw more bars, use more*  bar chart, add data 50 *blocks.*

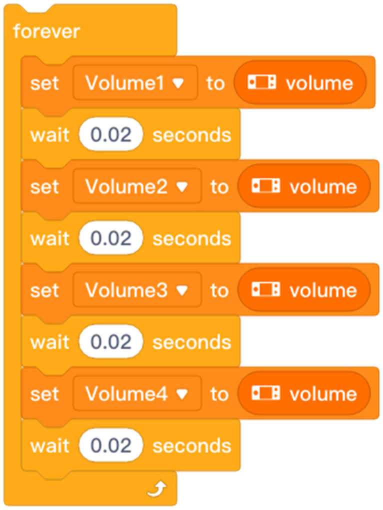
② *Use different colors to distinguish bars. If you set different bars to the same color, then all the data will be shown in the same bar. Look at the script below:*



```
set brush color ○
bar chart, add data 50
set brush color ●
bar chart, add data 50
set brush color ○
bar chart, add data 10
```

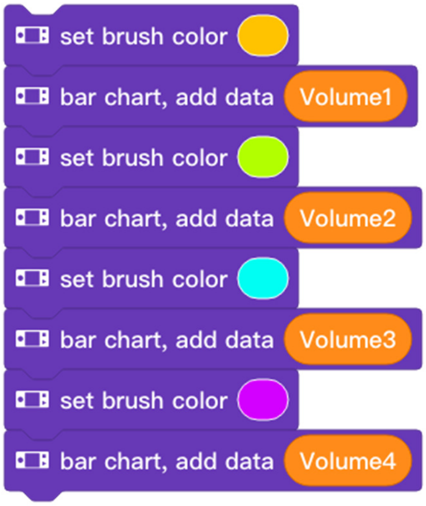
  - If we run the script above, a white bar and a red bar appear on the display. Every time this script is run, the white bar goes up to 50 first then falls to 10 while the red bar stays at 50 all the time.
  - To make volume bars with different heights, collect volume data at specific intervals.

  - To collect volume data, write the script below:



```
forever
    set Volume1 ▾ to volume
    wait 0.02 seconds
    set Volume2 ▾ to volume
    wait 0.02 seconds
    set Volume3 ▾ to volume
    wait 0.02 seconds
    set Volume4 ▾ to volume
    wait 0.02 seconds
```

- To continuously display sound levels, write a script as shown:



```
set brush color 🟡
bar chart, add data Volume1
set brush color 🟢
bar chart, add data Volume2
set brush color 🔵
bar chart, add data Volume3
set brush color 🟣
bar chart, add data Volume4
```

NOTE:

① Instruct the students to do the following:

(a) make a volume bar;

(b) make two bars with the same height;

(c) use variables to make bars with different heights;

(d) make four volume bars with different heights.

② To make the volume bars more dynamic, keep the intervals short.

*③ CyberPi starts drawing bars as soon as the scripts are executed. So, the lighting effect script and volume bar script should both start with the same **Event** block.*

Section 5 – Evaluate
[5 minutes]

Objective:

- Play some music and invite students to present their projects.

Procedures:

- Play music and invite students to present their projects.

  NOTE:

  ① *Divide students into groups and have them present the projects to their group members.*

  ② *Invite volunteers to present their projects in front of the whole class.*

Lesson 3

Remote Classroom Polling

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Describe the features of LAN, and broadcast and receive data through LAN
- Generate a bar chart based on the votes
- Analyze poll results and draw conclusions
- Use mind maps to analyze how to make a remote classroom polling system
- Write programs and complete the remote classroom polling project

Key Focus

- Analyzing data charts
- Sending and collecting data with LAN broadcasting
- Processing votes and generating a bar chart

Content Standards

Type	Indicator	Standard
K12 CS Framework	Practice 3-1	Identify complex, interdisciplinary, real-world problems that can be solved computationally.
K12 CS Framework	Practice 3-3	Evaluate whether it is appropriate and feasible to solve a problem computationally.
CSTA	2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
ISTE	5c	Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

Preparation

For the Teacher:

- A laptop or desktop with mBlock installed
- A CyberPi kit

For Students:

- Able to write programs with coding blocks
- Prior knowledge of CyberPi’s major features

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Use videos to bring up the voting and polling topic and introduce voting systems and LAN.
5 minutes	**Section 2 – Explore** • Prepare a few poll questions in which students may be interested. Encourage students to brainstorm the features that a voting system should have.
5 minutes	**Section 3 – Explain** • Instruct students to use mind maps to analyze the project and prepare to program.
20 minutes	**Section 4 – Elaborate** • Give students the freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Decide or have students decide the poll question. Give students time to try the voting system and analyze the result.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Use videos to bring up the voting and polling topic and introduce voting systems and LAN.

Procedures:

- Start the class by saying: Have you ever watched any competition shows? In these shows, a voting session usually follows the competitors' performances.
- **Ask:** During the live voting session, how did the audience vote? What about those watching the show at home? Could they participate in the session?
  Have students answer the questions.

- **Ask:** Do you know how to limit access to the voting session? Like in the voting session I mentioned just now, only the live audience can vote, and the home audience can't.
- Introduce LAN: One of the common ways is to connect the voting machines to the same private network. This kind of network is usually used within an area and hence, called Local Area Network (LAN). Unlike the Internet, a LAN is a closed network, which means only the devices connected to the same LAN can share resources and information.

Section 2 – Explore
[5 minutes]

Objective:

- Prepare a few poll questions in which students may be interested. Encourage students to brainstorm the features that a voting system should have.

Procedures:

- **Explain and ask:** In this lesson, we will use CyberPi to form a LAN and then run a poll. And only we, in this classroom, can vote. So, how do we make a voting system with CyberPi and mBlock?
  Give students time to discuss the question.
- **Conclude:** First, we need an initiator to start a poll. Then, the rest of us will be voters. When the voting session closes, the initiator counts the votes.

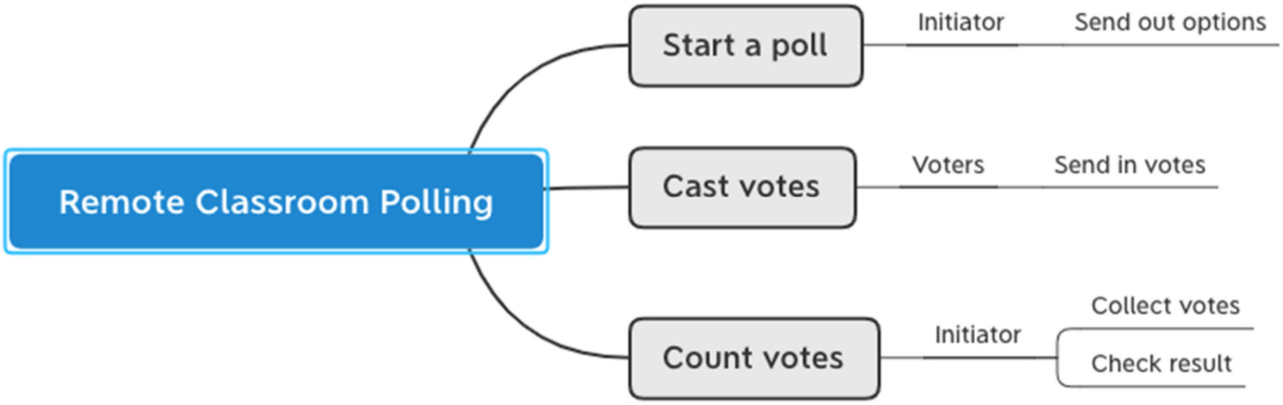
Section 3 – Explain
[5 minutes]

Objective:

- Instruct students to use mind maps to analyze the project and prepare to program.

Procedures:

- Demonstrate how remote classroom polling works.
- Analyze with students what features the remote classroom polling project should have and draw a mind map.



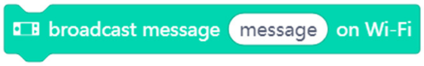
Section 4 – Elaborate
[20 minutes]

Objective:

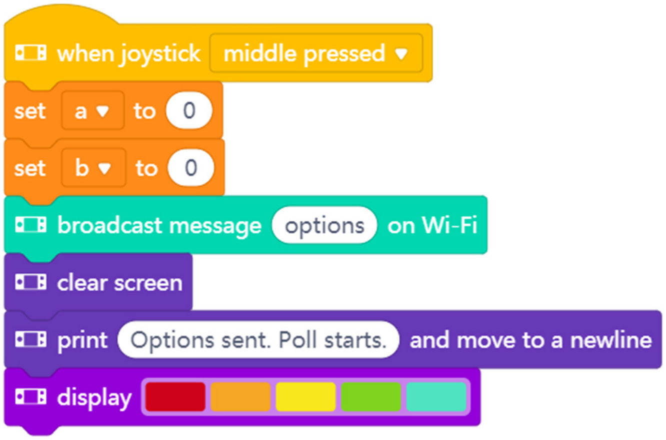
- Give students the freedom to explore further so that they can enhance their programming skills.

Procedures:

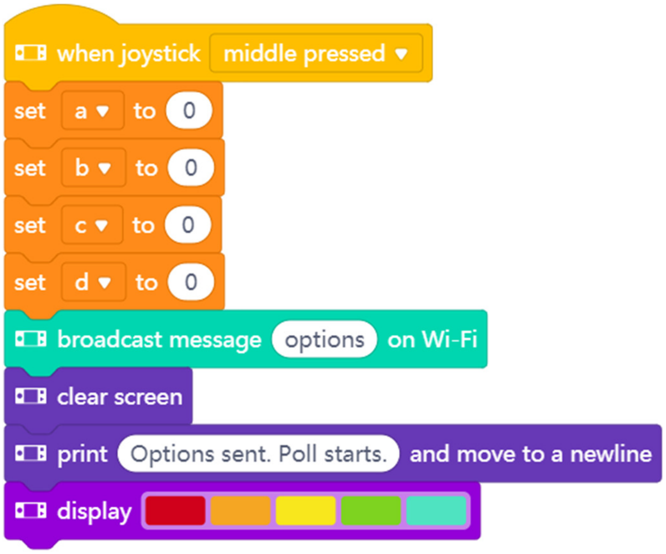
- **Explain:** We write programs for the initiator and the voter separately, so we'll need two CyberPi boards. In mBlock, select **Upload** mode before writing programs for the two boards.
- Walk students through the programs for the initiator and the voter respectively. To write programs for the initiator, select **CyberPi** in **Devices**, and select **CyberPi 2** to write programs for the voter.
- The initiator starts a poll:
    - To make multiple CyberPi boards communicate with each other, we need the **LAN** blocks.

Category	Coding block	Feature
LAN	 broadcast message message on Wi-Fi	This block sends a broadcast on LAN. Type the message you want to send in the slot.

-
    - Select **CyberPi** in **Devices** and write programs for the initiator.
    - Send out options:
        - Create two option variables to record the votes. The initiator CyberPi sends a LAN broadcast "options", displays a prompt "Options sent. Poll starts", and lights up indicators.



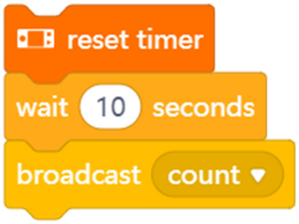
```
when joystick [middle pressed ▾]
set [a ▾] to (0)
set [b ▾] to (0)
broadcast message (options) on Wi-Fi
clear screen
print (Options sent. Poll starts.) and move to a newline
display 🟥🟧🟨🟩🟦
```



```
when joystick [middle pressed ▾]
set [a ▾] to (0)
set [b ▾] to (0)
set [c ▾] to (0)
set [d ▾] to (0)
broadcast message (options) on Wi-Fi
clear screen
print (Options sent. Poll starts.) and move to a newline
display 🟥🟧🟨🟩🟦
```

  - *NOTE: Create two variables to record the votes for the two options and set both variables to 0.*

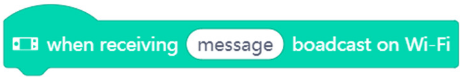
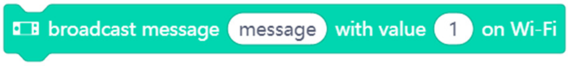
- Send out options/Set voting period
  - Apart from starting the poll, the initiator sets the voting period, so it needs to set a timer. Only during the voting period that is set up, voters can cast their votes. As the poll ends, the initiator starts counting the votes.



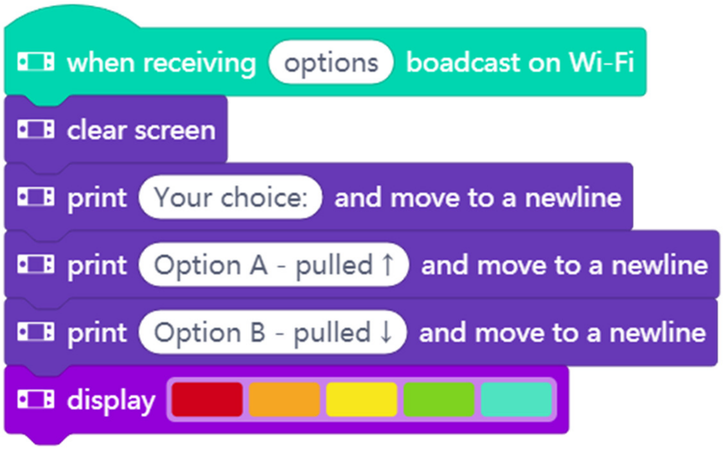
```
reset timer
wait (10) seconds
broadcast [count ▾]
```

  - *NOTE: The timer will be used to decide whether a voter casts a vote within the voting period.*

- Voters send in votes:
  - When receiving the broadcast "options", a voter starts to vote. During the voting session, voters send their votes on LAN through broadcasting. You'll need these blocks in your program:

Category	Coding block	Feature
LAN	 when receiving (message) boadcast on Wi-Fi	This block executes the script that follows it when receiving the broadcast. Make sure you type the correct broadcast in the slot.
	 broadcast message (message) with value (1) on Wi-Fi	This block sends a broadcast and a value. You can define the broadcast and the value. The value can be English letters or numbers.

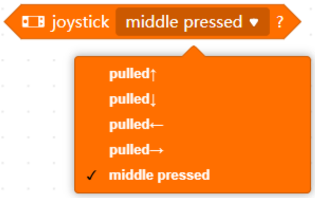
  - Select **CyberPi 2** in **Devices** and write programs for the voter.
  - Display options:
    - Type "options" in **when receiving ( ) broadcast on Wi-Fi**. Make sure the broadcast name is the same as the one in the initiator program. When receiving the broadcast, the voter CyberPi lights up indicators and displays the options and voting instructions.

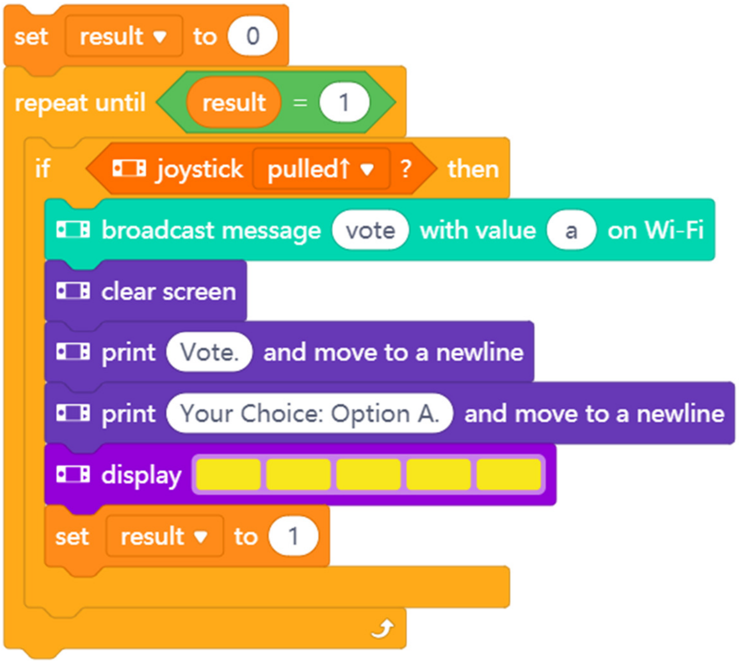


```
when receiving (options) boadcast on Wi-Fi
clear screen
print (Your choice:) and move to a newline
print (Option A - pulled ↑) and move to a newline
print (Option B - pulled ↓) and move to a newline
display 🟥🟧🟨🟩🟦
```

  - Cast votes:

- Make sure voters can cast only one vote by recording their voting status. The voter CyberPi sends a LAN broadcast "vote". When the vote is sent through, a prompt will appear on the screen of the voter CyberPi, and indicators will light up.
- Whether the joystick is pulled decides the voting status and the direction to which the joystick is pulled indicates the voter's choice. You'll need this block in your program:

Category	Coding block	Feature
Sensing	 joystick middle pressed ▾ ? pulled↑ pulled↓ pulled← pulled→ ✓ middle pressed	This Boolean block checks the action done to the joystick. It offers five options: **pulled ↑**, **pulled ↓**, **pulled ←**, **pulled →**, and **middle pressed**.



```
set result ▾ to 0
repeat until result = 1
    if joystick pulled↑ ▾ ? then
        broadcast message vote with value a on Wi-Fi
        clear screen
        print Vote. and move to a newline
        print Your Choice: Option A. and move to a newline
        display 🟨🟨🟨🟨🟨
        set result ▾ to 1
```

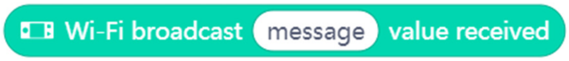
- *NOTE:*

  ① To make sure each voter can only cast one vote, create a variable "VotingStatus" to record whether a voter finishes voting. When the variable is 0, it means the voter hasn't voted yet. VotingStatus changes to 1 when the voter finishes voting.

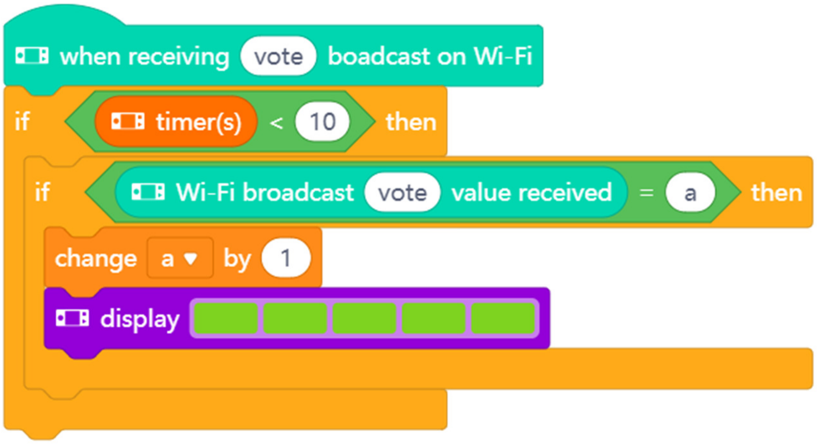
  ② The example program includes only one option. Complete the program that provides two voting options.

- Initiator collects votes:

- The initiator collects the votes via LAN broadcast. You’ll need this block to enable that:

Category	Coding block	Feature
LAN	 Wi-Fi broadcast (message) value received	This block receives the value sent with the broadcast. Make sure you type the correct broadcast in the slot.

- Select **CyberPi** in **Devices** and write another initiator program.
- Type "vote" in **when receiving ( ) broadcast on Wi-Fi**. Make sure the broadcast name is the same as the one sent from the voter program. When receiving the broadcast during the voting period, the initiator collects the votes and lights up indicators.



```
when receiving (vote) boadcast on Wi-Fi
if < (timer(s)) < (10) > then
    if < (Wi-Fi broadcast (vote) value received) = (a) > then
        change [a ▾] by (1)
        display [🟩🟩🟩🟩🟩]
```

- *NOTE:*

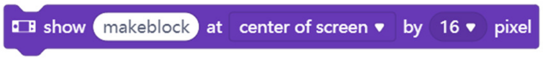
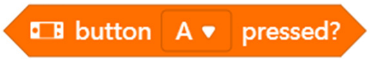
  ① *Include a timer in the script to check whether a vote is submitted within the voting period.*

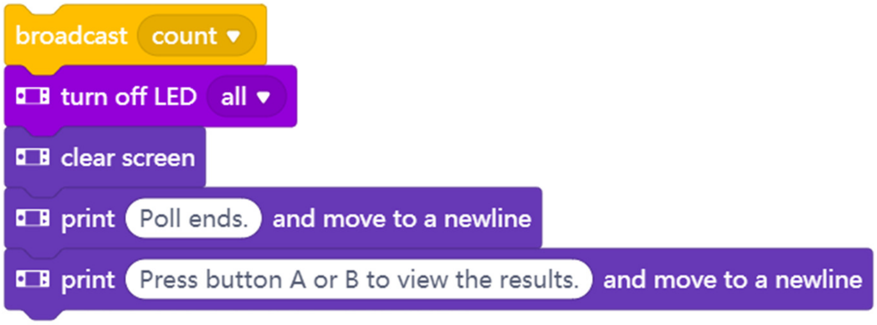
  ② *To count the votes, set a conditional statement to check the broadcast value, and increase the vote by 1 for the corresponding option based on the value received.*

  ③ *The example program counts the votes for option a. Complete the program that can count the votes for both options.*

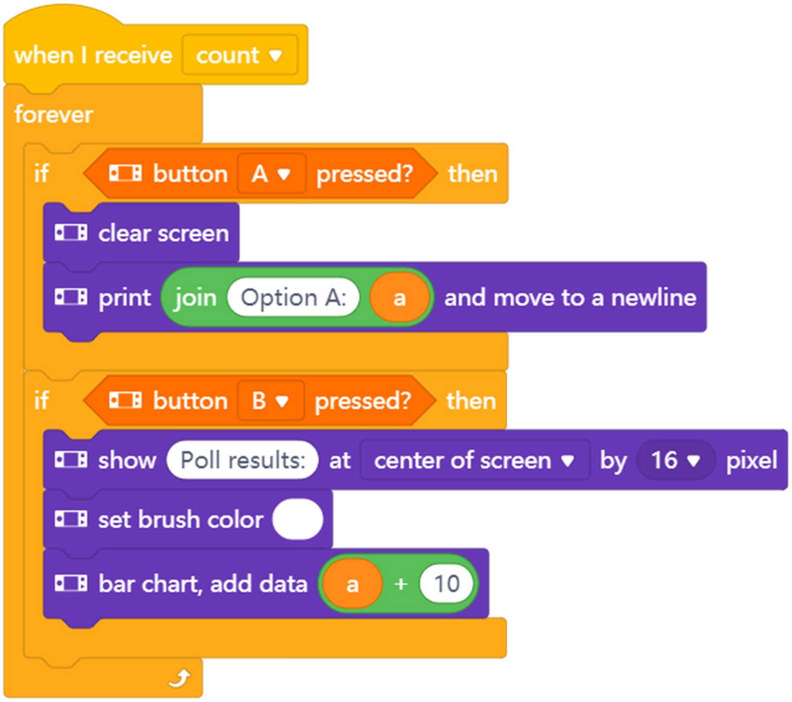
  ④ *If different CyberPi boards send through the same message via the same LAN at the same time, the receiver CyberPi will only receive the message once. That is if multiple voters select option A at the same time, then the initiator will count only one vote for Option A. However, the possibility of this incident is faint.*

- Initiator checks the polling result
    - You can display the polling result with text or a bar chart. And these blocks may help:

Category	Coding block	Feature
Display	 show makeblock at center of screen ▾ by 16 ▾ pixel	This block makes CyberPi display text and specifies the location and size of the text. It includes nine locations and four font sizes.
Sensing	 button A ▾ pressed?	This Boolean block detects whether button A or B is pressed.



```
broadcast count ▾
turn off LED all ▾
clear screen
print Poll ends. and move to a newline
print Press button A or B to view the results. and move to a newline
```



```
when I receive count ▾
forever
    if button A ▾ pressed? then
        clear screen
        print join Option A: a and move to a newline
    if button B ▾ pressed? then
        show Poll results: at center of screen ▾ by 16 ▾ pixel
        set brush color
        bar chart, add data a + 10
```

- ○ *NOTE:*

  ① The example program displays the vote tally for option a. Complete the program that displays the vote tally for both options.

  ② Set different colors for different bars when visualizing the result with a bar chart.

  ③ If the number of voters is small, add a base number to the vote tally of each option. For example, a + 10*. In this way, the bar chart will be clearer.*

- Allow students time to test their programs and then invite them to describe the bar chart and draw conclusions.

Section 5 – Evaluate
[5 minutes]

Objective:

- Decide or have students decide the poll question. Give students time to try the voting system and analyze the result.

Procedures:

- Initiate a classroom poll and invite students to vote. (Possible subject: favorite song/movie star/singer/game, etc.)
- When the poll is done, invite students to analyze the result data chart. For example, have students figure out which is the most popular song among the class and why it is so popular.

Lesson 4
Houseplant Care

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Explain the relationship between plants and water and identify how much water different plants need
- Simulate sunlight and monitor the moisture of plants
- Use mind maps to analyze how a plant monitoring device works

Key Focus

- Use sensors to measure and monitor the moisture of plants
- Simulate sunlight and water loss using sensors and send alarms when water is insufficient or too much

Content Standards

Type	Indicator	Standard
K12 CS Framework	Practice 3-1	Identify complex, interdisciplinary, real-world problems that can be solved computationally.
K12 CS Framework	Practice 3-3	Evaluate whether it is appropriate and feasible to solve a problem computationally.
CSTA	2-IC-20	Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options.
CSTA	2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
ISTE	5c	Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

Preparation

For the Teacher:

- A laptop or desktop with mBlock installed
- A CyberPi kit

For Students:

- Able to write programs with coding blocks
- Prior knowledge of CyberPi's major features

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Before students design plant monitoring devices, help them learn about plants and the purpose of the houseplant care project.
5 minutes	**Section 2 – Explore** • Introduce the application of a plant monitoring device. Encourage students to think about how to design this project. Enhance students' independent thinking skills.
5 minutes	**Section 3 – Explain** • Make students analyze the Plant Monitoring Device project's functionalities with mind maps. Ask them to consider what the plant monitoring device could perform.
20 minutes	**Section 4 – Elaborate** • Instruct students to design the Plant Monitoring Device project. • Have students program the different functions of the plant monitoring device.
5 minutes	**Section 5 – Evaluate** • Have students further explore what they've learned based on this project.

☰ Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Before students design plant monitoring devices, help them learn about plants and the purpose of the houseplant care project.

Procedures:

- Introduce the topic of this lesson.
    - Say: Plants can clean air, relax our minds and eyes, and add color to the environment. All life needs water. We need water, so do plants.
    - Play a video demonstration to help students learn more about plants.
    - Summarize: Water is essential in plant's healthy growth, photosynthesis, and transpiration. Plants have different features, and hence, different water needs. For example, cacti, covered in spines, dry out very slowly, so we can water them once a month. Some plants, such as Chinese money plants (i.e. Pilea Peperomioides) with flat, thin, big leaves, need much water, so we'd better water them every day. Most plants, such as chrysanthemums and morning glories, only need watering once a week
    - Ask: What will happen if we water our plants too frequently or forget to water them?
    - Have students think about and discuss the above question.
    - Summarize: Plants will dry out and wither if we forget to water them. But if we water them too often, they may die of overwatering. We should be aware that overwatering wastes water, especially when it comes to watering many plants, like during agricultural irrigation.

Section 2 – Explore
[5 minutes]

Objective:

- Introduce the application of a plant monitoring device. Encourage students to think about how to design this project. Enhance students' independent thinking skills.

Procedures:

- Introduce the design challenge of this lesson: To design a plant monitoring device to remind plant lovers to water their plants appropriately.
  - Ask: What features do you think a plant monitoring device should contain?
  - **Possible answers:** A plant monitoring device could be able to:
    - Detect whether a plant has a moisture problem; and
    - Remind the user to take care of the plant when it has a moisture problem.

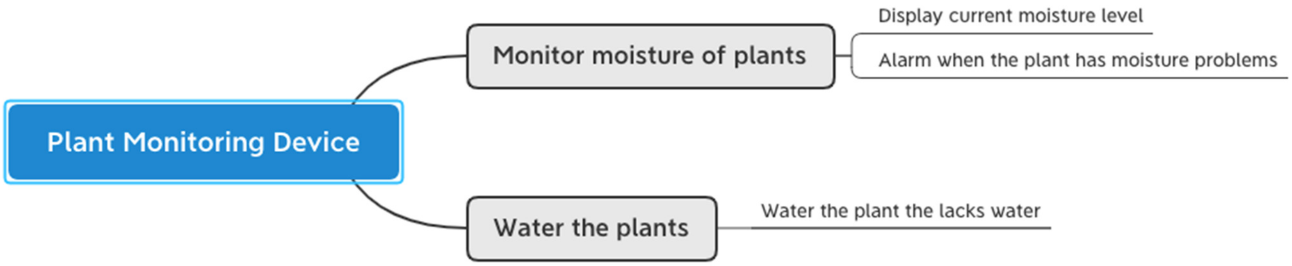
Section 3 – Explain
[5 minutes]

Objective:

- Make students analyze the Plant Monitoring Device project's functionalities with mind maps. Ask them to consider what the plant monitoring device could perform.

Procedures:

- Demonstrate the desired outcomes of the Plant Monitoring Device project.
- Instruct students to identify the functions and applications of the plant monitoring device by means of the mind map.


Plant Monitoring Device
Monitor moisture of plants
Display current moisture level
Alarm when the plant has moisture problems
Water the plants
Water the plant the lacks water

Section 4 – Elaborate
[20 minutes]

Objectives:

- Instruct students to design the Plant Monitoring Device project.
- Have students program different functions of the plant monitoring device.

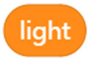
Procedures:

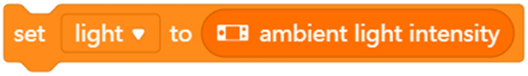
- Prepare to write Python programs.
    - Have students get their CyberPi devices ready, connect it to mBlock, and switch to **Live** mode in mBlock.
    - Distribute the example program to students and ask them to open it.
    - Observe the three sprites on the stage and analyze their possible features.

      [Note: Assume the three plants are given the same amount of sunlight in this project.]

Plant Sprite	Feature
	Cactus7 • Low demand for water • Slowly dries out
	Pot plant12 • Medium demand for water • Dries out at a medium speed
	Pot plant1 • High demand for water • Quickly dries out

- Instruct students to write programs based on the mind map for project design.
- Instruct students to program the project. Make it display current moisture level.
    - **Simulate the water loss process:** Use the light sensor to simulate the light. When the light is intense, the plant loses water faster. Remind students that plants have different water loss speeds, so they need to set up the speed accordingly.
        - Create a variable **light** to store the data returned by the light sensor.


light

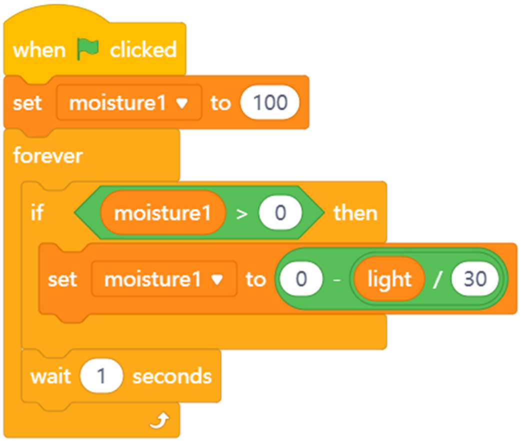
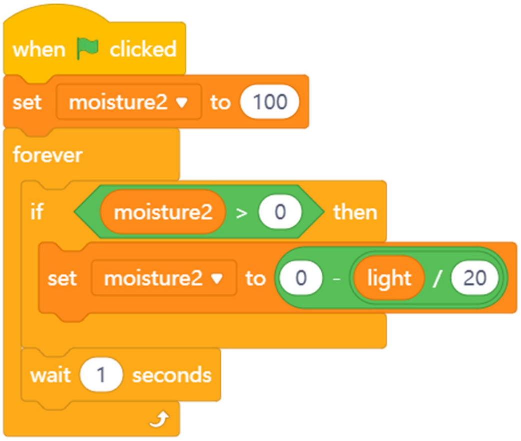
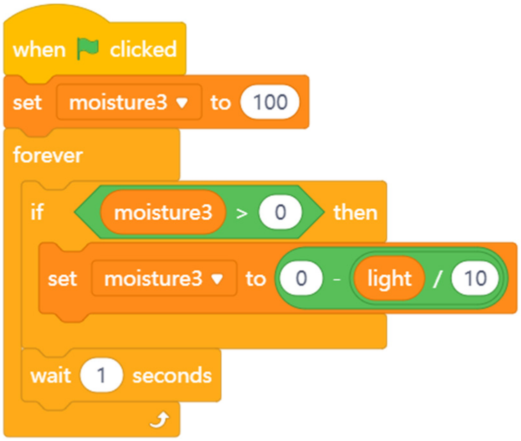

set light ▾ to ambient light intensity

- Create variables **moisture1**, **moisture2**, and **moisture3**. Set different variables to represent the moisture level of the three plants:

Plant Sprite	Variable
	Cactus7 moisture1
	Pot plant12 moisture2
	Pot plant1 moisture3

- Simulate the water loss of the plants:

Plant Sprite	Script
	Plant “Cactus7” slowly dries out:

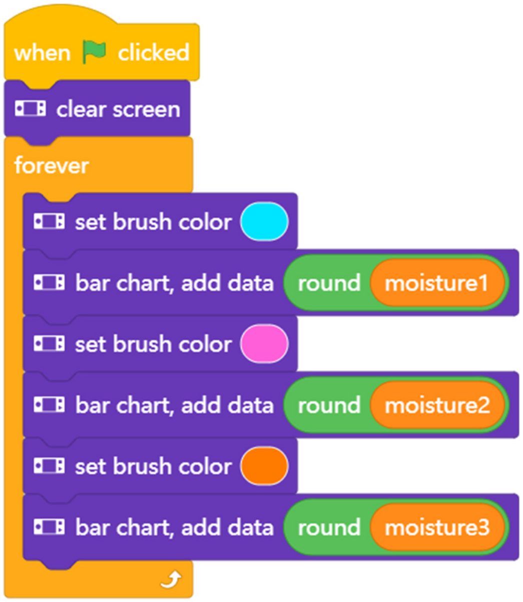
	 when ⚑ clicked set moisture1 ▾ to 100 forever if moisture1 > 0 then set moisture1 ▾ to 0 - light / 30 wait 1 seconds
	Plant “Pot plant12” out at medium speed:  when ⚑ clicked set moisture2 ▾ to 100 forever if moisture2 > 0 then set moisture2 ▾ to 0 - light / 20 wait 1 seconds
	Plant “Pot plant1” quickly dries out:  when ⚑ clicked set moisture3 ▾ to 100 forever if moisture3 > 0 then set moisture3 ▾ to 0 - light / 10 wait 1 seconds

Tips for coding:

- Divide the variables light by different numbers to simulate the plants' drying out process.

Plant	Divisor	Drought resistance	Water loss speed
Cactus7	30	High	Slow
Pot plant12	20	Medium	Medium
Pot plant1	10	Low	Fast

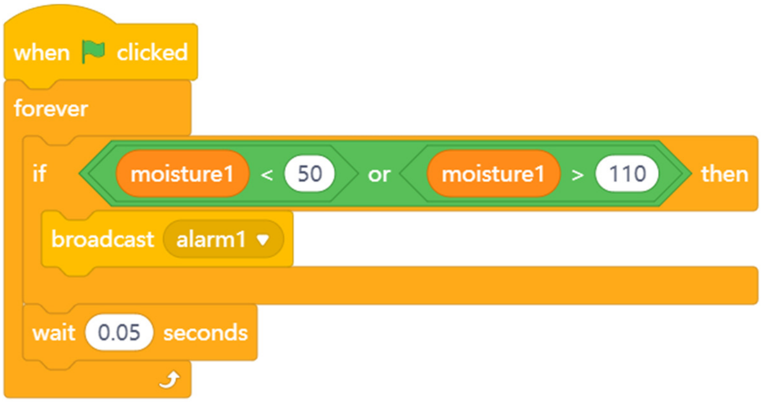
        - Set the condition "moisture is greater than 0" to avoid the variables going negative.
        - Use the wait block to prevent the moisture variables from decreasing too fast.
    - **Display moisture level:** CyberPi's screen can display a bar chart to represent the moisture level of the three plants



```
when ⚑ clicked
clear screen
forever
    set brush color ●
    bar chart, add data round moisture1
    set brush color ●
    bar chart, add data round moisture2
    set brush color ●
    bar chart, add data round moisture3
```

        - Use different colors to represent the indicators for different plants.
        - Round off the moisture values with the round block because CyberPi can only display integers.

- Make the project send alarms when plants have a moisture problem.
    - **Detect moisture problems:** The program decides whether to send an alarm.



```
when 🏴 clicked
forever
    if < (moisture1) < (50) > or < (moisture1) > (110) > then
        broadcast (alarm1 ▾)
    wait (0.05) seconds
```

- Ask students to complete the scripts for Plant “Pot plant12” and Plant “Pot plant1” in reference to the above example.
- Set the two conditions:

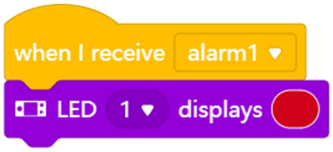
  If the moisture is under 50, it means the plant lacks water;

  If the moisture is over 110, it means the plant is overwatered.

  When one of the conditions is true, the program sends the alarm.
- Remind students that they can change the numbers (here 50 and 110) based on their design.

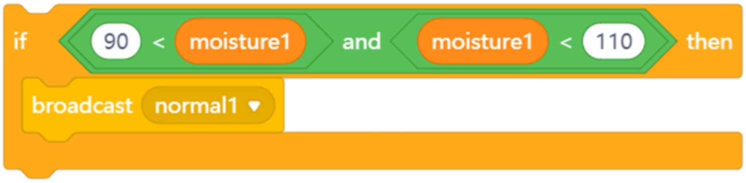
○ **Raise alarms:** When the plant has a moisture problem, a light indicator lights up.

- **Example of Plant ‘Cactus7’ Indicator:**

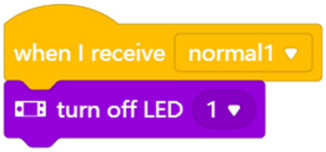


```
when I receive (alarm1 ▾)
LED (1 ▾) displays (●)
```

- Ask students to complete the scripts for Plant “Pot plant12” Indicator and Plant “Pot plant1” Indicator in reference to the above example.
- However, when the plant's moisture returns to the normal range, the alarm should stop.



```
if < (90) < (moisture1) > and < (moisture1) < (110) > then
    broadcast (normal1 ▾)
```

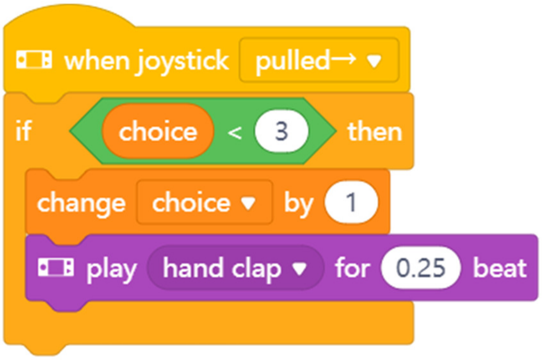


```
when I receive (normal1 ▾)
turn off LED (1 ▾)
```

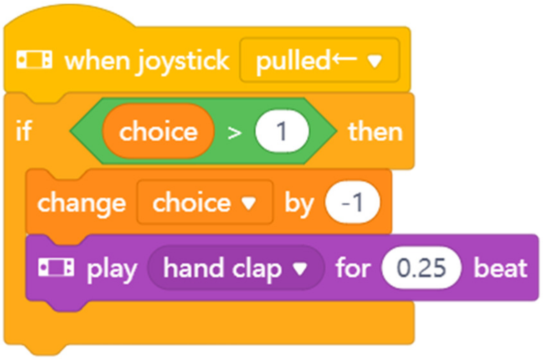
- Ask students to complete the scripts for Plant “Pot plant12” Indicator and Plant “Pot plant1” Indicator in reference to the above example.

    - Remind students that they can change the numbers (here 90 and 110) based on their design.

- Program the watering process.
  - **Use the joystick to select a plant:** Create another variable and name it: **choice**. Use this variable to select a plant.



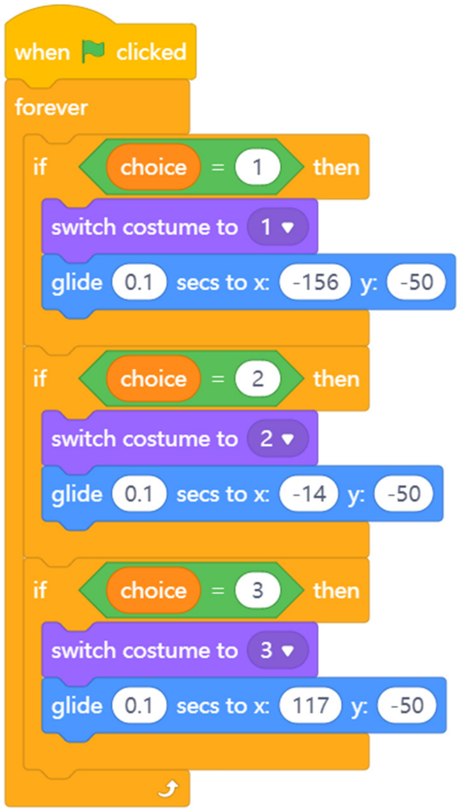
```
when joystick pulled→
if choice < 3 then
    change choice by 1
    play hand clap for 0.25 beat
```



```
when joystick pulled←
if choice > 1 then
    change choice by -1
    play hand clap for 0.25 beat
```

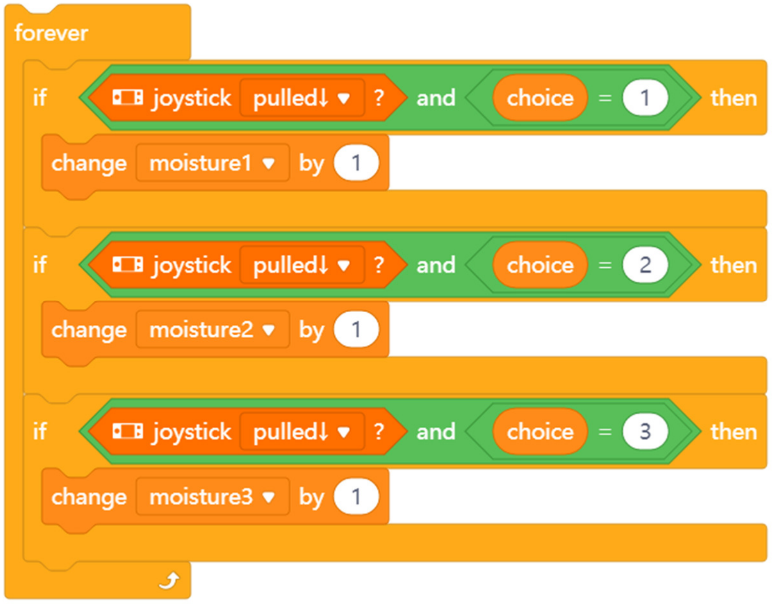
    - The variable **choice** can increase only when it's less than 3 and decrease only when itis greater than 1. Use this setting to limit the variable between 1 and 3.
    - When the value of **choice** equals 1, the first plant is selected. When the value of **choice** equals 2, the second plant is selected. When the value of **choice** equals 3, the third plant is selected.

  - Add a new sprite “Diamond3”. Use it as an arrow that will appear under the selected plant on the stage. Modify the sprite’s costumes to differentiate the arrows. Program this sprite as follows:



```
when 🏴 clicked
forever
    if choice = 1 then
        switch costume to 1 ▾
        glide 0.1 secs to x: -156 y: -50
    if choice = 2 then
        switch costume to 2 ▾
        glide 0.1 secs to x: -14 y: -50
    if choice = 3 then
        switch costume to 3 ▾
        glide 0.1 secs to x: 117 y: -50
    ↺
```

- ○ **Water the plants:** After you select a plant, pull the joystick down to water it.



```
forever
    if joystick pulled↓ ▾ ? and choice = 1 then
        change moisture1 ▾ by 1
    if joystick pulled↓ ▾ ? and choice = 2 then
        change moisture2 ▾ by 1
    if joystick pulled↓ ▾ ? and choice = 3 then
        change moisture3 ▾ by 1
    ↺
```

Section 5 – Evaluate
[5 minutes]

Objective:

- Have students further explore what they've learned based on this project.

Procedures:

- Invite volunteer students to present their projects. Ask students to give feedback on each other's projects and think about how to improve the projects.
- Have students introduce a plant (its name, features, and how to take care of it). Encourage them to do online research about the growth characteristics of the plant.

Lesson 5

Smart Notifier

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

Objectives

By the end of class, students will be able to:

- Use CyberPi to measure light intensity and shaking status.
- Define notifications for specific scenarios based on features and requirements.
- Use mind maps to analyze how to create a smart notifier.
- Program a Smart Notifier which features multiple functions and modes.

Key Focus

- Defining notifications for specific scenarios based on the features and requirements.
- Programming the smart notifier to send notifications for specific scenarios.

Content Standards

Type	Indicator	Standard
K12 CS Framework	Practice 3-1	Identify complex, interdisciplinary, real-world problems that can be solved computationally.
K12 CS Framework	Practice 3-3	Evaluate whether it is appropriate and feasible to solve a problem computationally.
K12 CS Framework	Practice 5-2	Create a computational artifact for practical intent, personal expression, or to address a societal issue.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.

Preparation

For the Teacher:

- A laptop or desktop with mBlock installed
- A CyberPi kit

For Students:

- Basic knowledge of how to use sensors and CyberPi

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Guide students in thinking about their daily habits and how these habits impact their health. Have them consider using AI technologies to solve problems.
5 minutes	**Section 2 – Explore** • Introduce students to some scenarios and have them think about how to design CyberPi to send notifications under these scenarios.
5 minutes	**Section 3 – Explain** • Based on mind maps, have students build an overall picture of the project, preparing them for the later programming session.
20 minutes	**Section 4 – Elaborate** • Give students the freedom to explore further, so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • To enhance students' communication skills and understanding of their projects, ask them to share their projects in front of the class. In this way, students will be likely to pay more attention to their health.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Guide students in thinking about their daily habits and how these habits impact their health. Have them consider using AI technologies to solve problems.

Procedures:

- **Ask:** The COVID-19 pandemic has provoked a global crisis that had a deep impact on our everyday life and learning. Schools are ordered to close, and people must stay at home. The pandemic makes us realize the importance of health. Think about it! Do you have any daily habits that may be bad for your health?
- **Summarize:** Not everyone cares enough about his or her health. Lots of people have some unhealthy habits, for instance, using phones while lying down, sitting, or standing with a curved back and slumped shoulders, and staying up late. In a tech-driven world, we could rely on modern technologies to track our health conditions. For example, we use wearables to count steps, and track our heart rates and sleep conditions through night.

Section 2 – Explore
[5 minutes]

Objective:

- Introduce students to some scenarios and have them think about how to design CyberPi to send notifications in these scenarios.

Procedures:

- Have students discuss: In school, we do setting-up and eye exercises to relax our body and protect eyes. But when at home, many of you will sit there for a long time, watching TV, playing video games or on their phones. They don't move quite often. Sometimes you are just so into reading that you fail to notice the light intensity level. Can you think of any ways to solve these problems?
- **Summarize:** We can design a product to monitor people's conditions and give alerts. For example, if people sit too long, the product will remind them to stand up and relax their body; if it's dark, the product will remind them to turn on the lights.

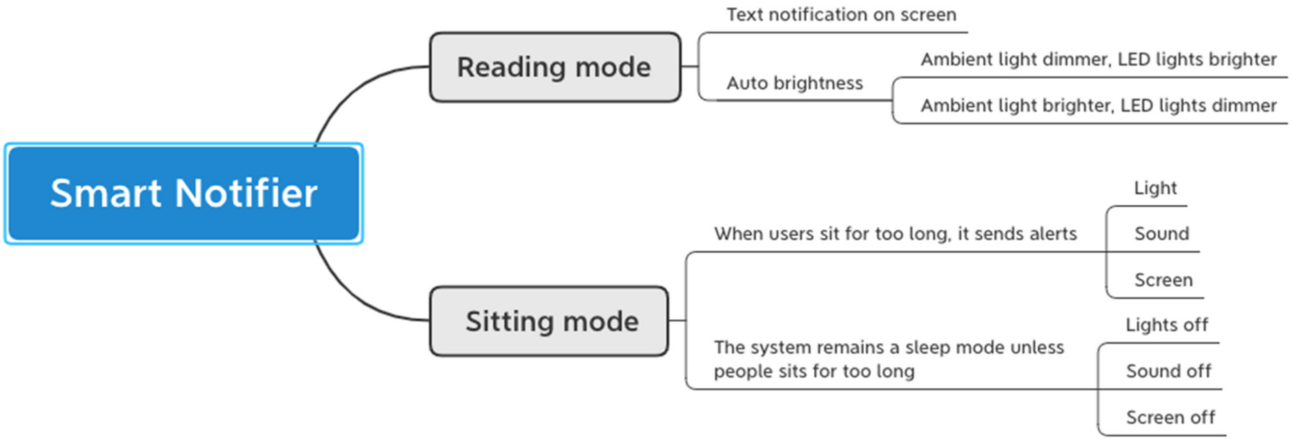
Section 3 – Explain
[5 minutes]

Objective:

- Based on mind maps, let students create an overall picture of the project, preparing them for the later programming session.

Procedures:

- **Introduce:** We can use CyberPi to design a smart notification system which features 2 modes – "Reading Mode" and "Sitting Mode". You can change the mode to let the system give specific notifications.
- Show students how the smart notification system works.
- Work with students to analyze what features their smart notifier must have and draw a mind map.


Smart Notifier
Reading mode
Text notification on screen
Auto brightness
Ambient light dimmer, LED lights brighter
Ambient light brighter, LED lights dimmer
Sitting mode
When users sit for too long, it sends alerts
Light
Sound
Screen
The system remains a sleep mode unless people sits for too long
Lights off
Sound off
Screen off

Section 4 – Elaborate

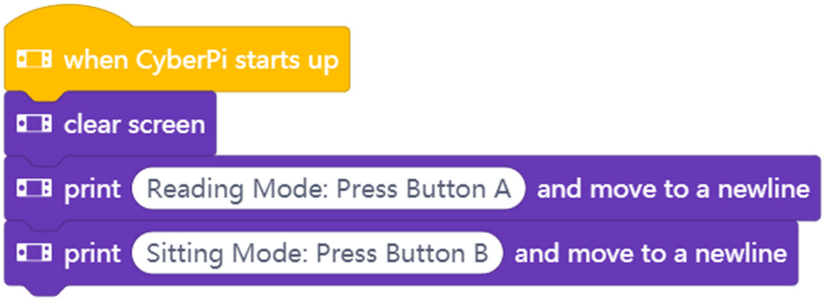
[20 minutes]

Objective:

- Give students the freedom to explore further so that they can enhance their programming skills.

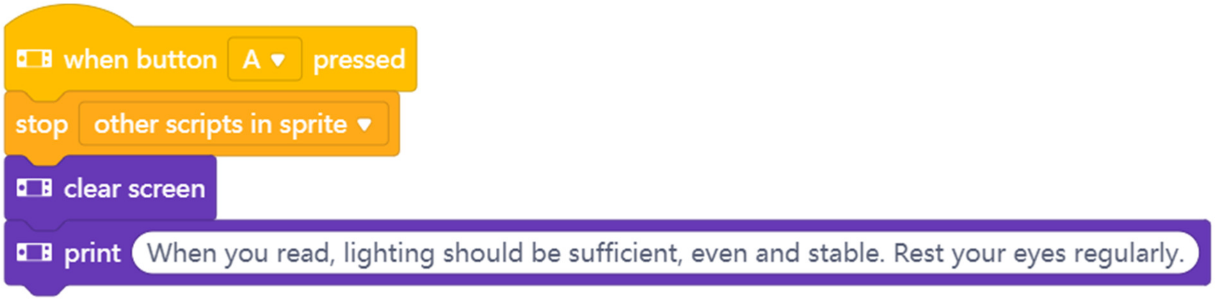
Procedures:

- Prepare a CyberPi, connect it to mBlock, and switch to **Live** mode.
- Design the startup interface, prompting users to select a mode.



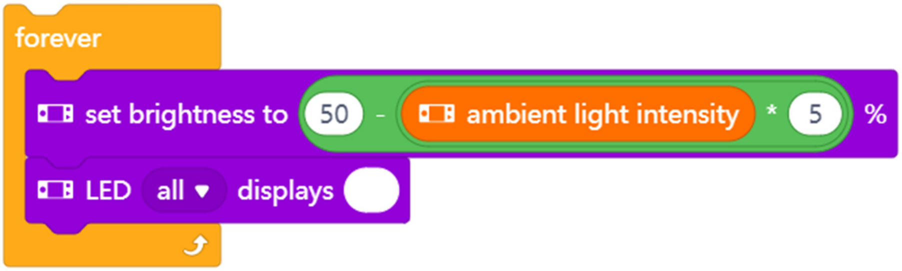
```
when CyberPi starts up
clear screen
print (Reading Mode: Press Button A) and move to a newline
print (Sitting Mode: Press Button B) and move to a newline
```

- Reading mode:
  - **Notification text on screen:** The program for the other mode stops when one of the modes is selected.



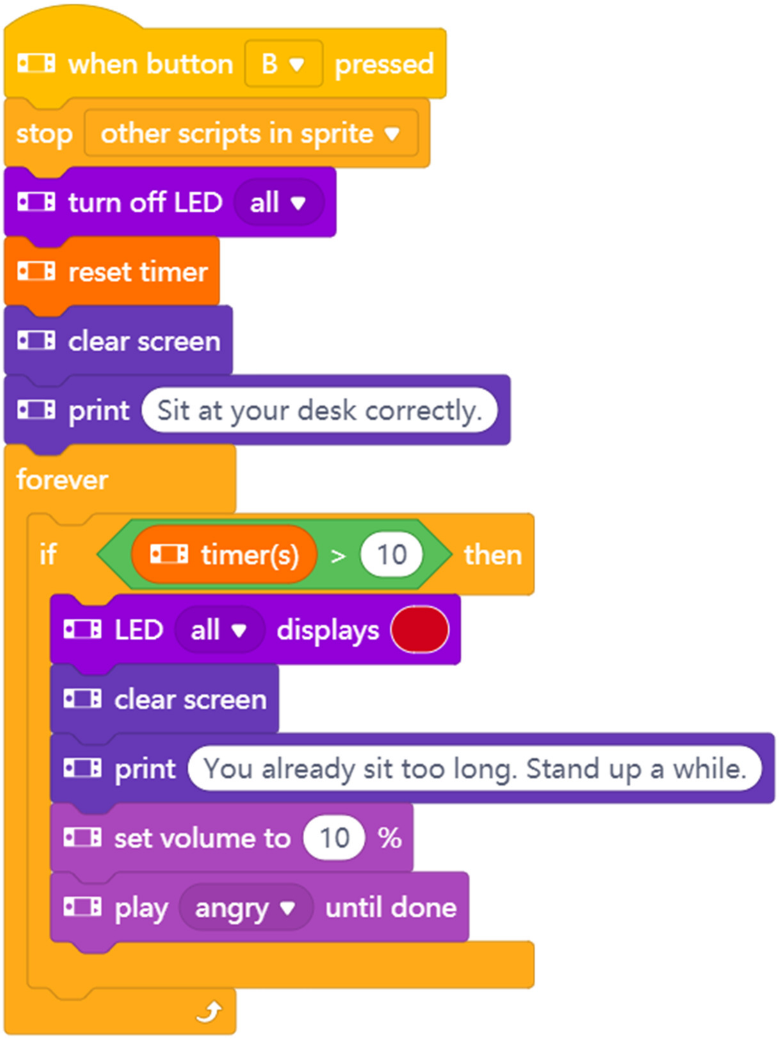
```
when button (A ▾) pressed
stop (other scripts in sprite ▾)
clear screen
print (When you read, lighting should be sufficient, even and stable. Rest your eyes regularly.)
```

- **Auto brightness:**
  - Program the light sensor. Set the brightness of LED lights to "100 – ambient light intensity". In this way, when the ambient light gets dimmer, the LED lights turn brighter; when the ambient light gets brighter, LED lights turn dimmer.



```
forever
    set brightness to (50 - (ambient light intensity * 5)) %
    LED all ▾ displays ○
↺
```

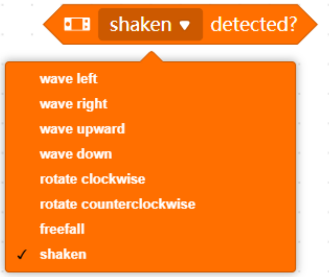
- Sitting mode:
- When you sit too long, it sends alerts:

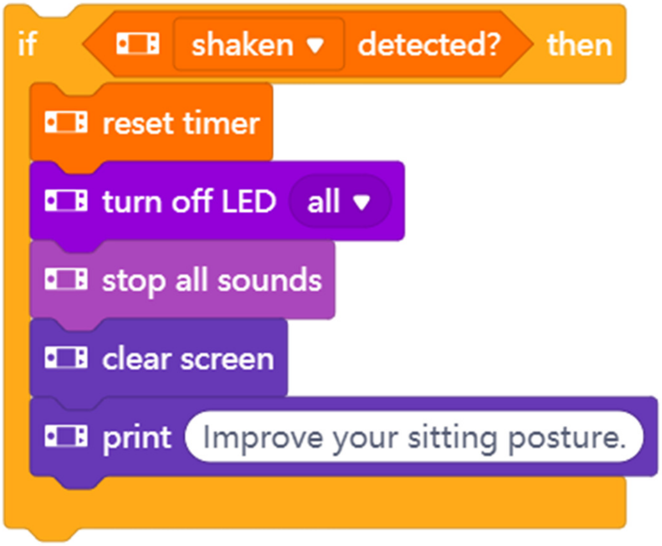


```
when button B ▾ pressed
stop other scripts in sprite ▾
turn off LED all ▾
reset timer
clear screen
print Sit at your desk correctly.
forever
    if timer(s) > 10 then
        LED all ▾ displays ●
        clear screen
        print You already sit too long. Stand up a while.
        set volume to 10 %
        play angry ▾ until done
↺
```

  - The program for the other mode stops when one of the modes is selected.
  - Options: Lights, Sounds, Texts
- The system remains a sleep mode unless people sit too long.
  - The system maintains a sleep mode unless people sit too long. Program the device to determine whether people are sitting there too long or not, based on the shaking status.

- Introduce the coding block in the table.

Category	Coding block	Feature
Motion Sensing	 shaken ▾ detected? wave left wave right wave upward wave down rotate clockwise rotate counterclockwise freefall ✓ shaken	Use this block to detect trigger actions. When a specified action is detected, the condition is met. There're eight actions, such as "wave left", "waved right" and more.



```
if shaken ▾ detected? then
    reset timer
    turn off LED all ▾
    stop all sounds
    clear screen
    print Improve your sitting posture.
```

- The timer is reset once shaking is detected.
- Repeat this piece of code.

Section 5 – Evaluate
[5 minutes]

Objective:

- To enhance students' communication skills and understanding of their projects, ask them to share their projects in front of the class. In this way, students will be likely to pay more attention to their health.

Procedures:

- Invite 2 or 3 students to share their smart notifier in front of the class and have them explain what's special about their project.
- **Have students discuss:** Can you think of any other unhealthy habits? Can you think of any methods to improve your health?

Lesson 6

Parcel Locker

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- List the ways to receive parcels and describe the significance of parcel lockers.
- Explain how a parcel locker works and write programs to send pickup codes and collect parcels with a pickup code.
- Use mind maps to analyze the features of a parcel locker.
- Apply variables in programs to complete a parcel locker system.

Key Focus

- Using variables to send and verify pickup codes

Content Standards

Type	Indicator	Standard
K12 CS Framework	Practice 3-1	Identify complex, interdisciplinary, real-world problems that can be solved computationally.
K12 CS Framework	Practice 5-2	Create a computational artifact for practical intent, personal expression, or to address a societal issue.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
CSTA	2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.

Preparation

For the Teacher:

- A laptop or desktop with mBlock 5 installed
- A CyberPi kit

For Students:

- Block-based coding skills
- Knowledge of how to use variables

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • List the ways to collect parcels and explain the application of parcel lockers.
5 minutes	**Section 2 – Explore** • Describe the features of a parcel locker and encourage the students to brainstorm how to develop the features.
5 minutes	**Section 3 – Explain** • Instruct students to use mind maps to analyze the project and prepare them for the Elaborate session.
20 minutes	**Section 4 – Elaborate** • Give students the freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Allow students time to test their projects. Have them think about the relationship between parcel lockers and data as well as the use of data in everyday life.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- List the ways to collect parcels and explain the application of parcel lockers.

Procedures:

- **Ask:** How do you receive parcels?
- **Give your answer:** We can receive a parcel at home, pick it up at a specified collection point, such as a guard booth and a store, or collect it from a parcel locker nearby.
- **Ask:** If nobody is at home, which will be your choice? Why?
- **Summarize:** When nobody is at home, having the parcel delivered to a parcel locker will be the best choice. Think about it. If the courier just drops the parcel at the door, it may get lost. If the parcel is sent to a collection point like a store, other people may mistakenly take it. A parcel locker can avoid such problems because only the recipient with the one-off pickup code can open the locker into which the courier put the parcel. Also, a parcel locker works 24/7.

Section 2 – Explore
[5 minutes]

Objective:

- Describe the features of a parcel locker and encourage the students to brainstorm how to develop the features.

Procedures:

- **Ask:** After the courier puts a parcel into the parcel locker, how does the recipient know the parcel arrived and is ready for pick up?
- **Give your answer:** The recipient will receive a text message carrying a pickup code. Then the recipient needs to go to the parcel locker and enter the pick-up code. A door will open if the recipient enters the code correctly.

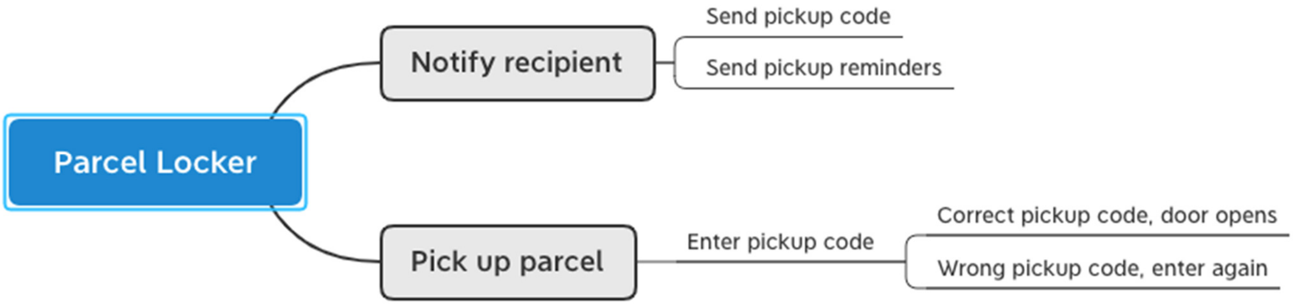
Section 3 – Explain
[5 minutes]

Objective:

- Instruct students to use mind maps to analyze the project and prepare them for the Elaborate session.

Procedures:

- Demonstrate how the parcel locker works.
- Work with students to analyze what features a parcel locker should have and draw a mind map.



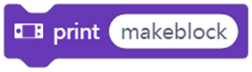
Section 4 – Elaborate
[20 minutes]

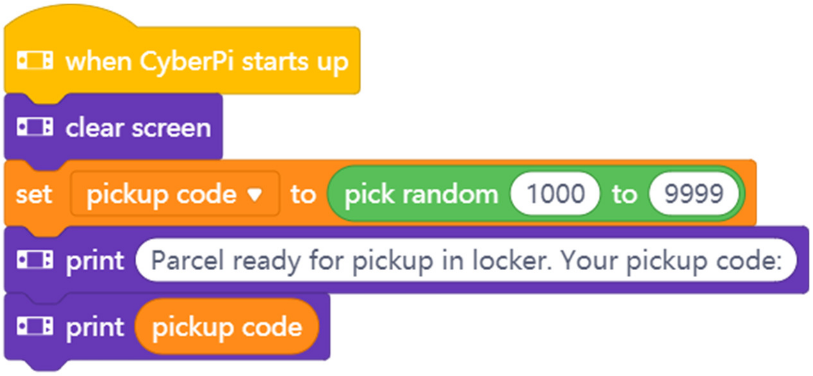
Objective:

- Give students the freedom to explore further so that they can enhance their programming skills

Procedures:

- Connect CyberPi to mBlock.
- Open the preset program file.
- Notify recipient:
    - **Send pick-up code:** When a parcel is delivered to a parcel locker, a pickup code is sent to CyberPi. Create a variable "Pick-up Code" to store the code, making it easier for the program to check whether the input code is correct.
    - Use this block to display the parcel information:

Category	Coding block	Feature
Display	 print makeblock	This block displays a message on CyberPi's screen. Type in the text you want to display.



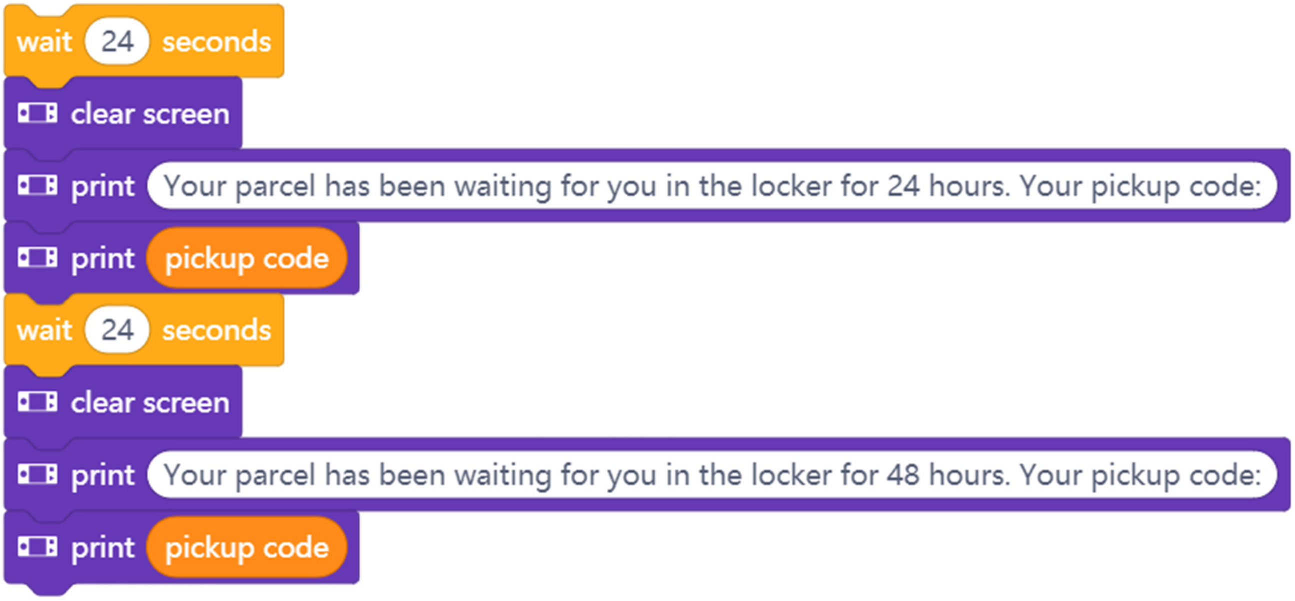
```
when CyberPi starts up
clear screen
set pickup code ▾ to pick random 1000 to 9999
print Parcel ready for pickup in locker. Your pickup code:
print pickup code
```

- *NOTE:*

① Generate pseudorandom numbers as pick-up codes. The example program generates 4-digit codes. If you want 5-digit codes, limit the range to 10,000 to 99,999.

*② Compare **print ( )** and **print ( ) and move to a new line**.*

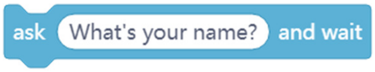
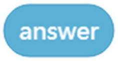
- **Pickup reminder**: If the parcel isn’t collected after a certain period, a pick-up reminder will be sent to the recipient. This setting improves the efficiency of the parcel locker. Set two timeframes so that a reminder will be sent if a parcel isnot collected after 24 hours and 48 hours.



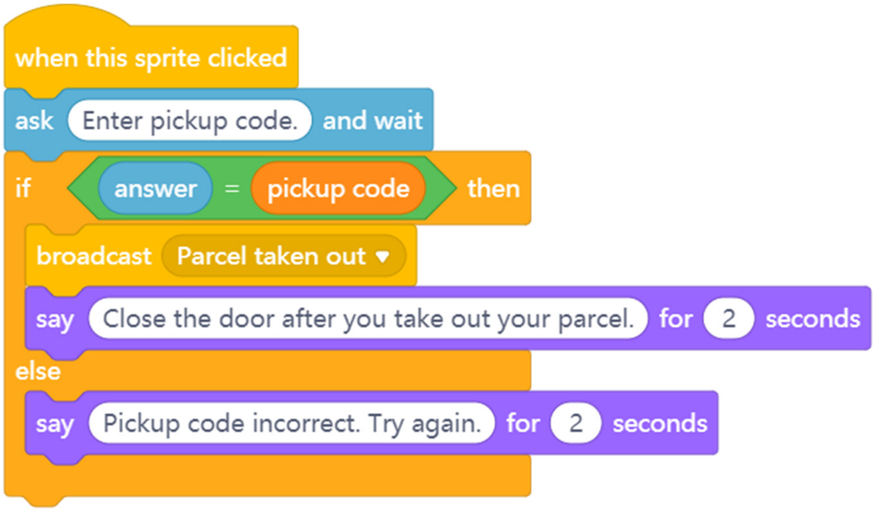
```
wait (24) seconds
clear screen
print (Your parcel has been waiting for you in the locker for 24 hours. Your pickup code:)
print (pickup code)
wait (24) seconds
clear screen
print (Your parcel has been waiting for you in the locker for 48 hours. Your pickup code:)
print (pickup code)
```

- *NOTE: To save time, use 24 seconds to represent 24 hours in this project.*

- Pick up parcel

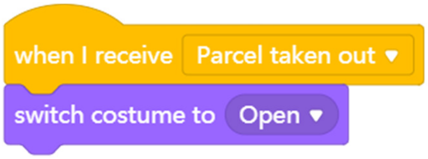
Category	Coding block	Feature
● Sensing	 ask (What's your name?) and wait	A sensing block makes a sprite ask a question on the stage and waits for you to input an answer.
	 answer	This block stores the answer you input.

- After receiving the pickup code, tap the "Pick Up" button and enter the code.



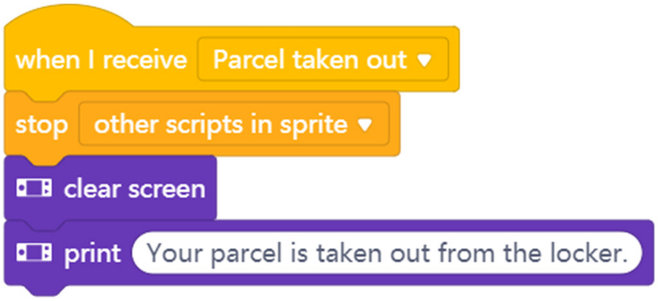
```
when this sprite clicked
ask (Enter pickup code.) and wait
if <(answer) = (pickup code)> then
    broadcast (Parcel taken out ▾)
    say (Close the door after you take out your parcel.) for (2) seconds
else
    say (Pickup code incorrect. Try again.) for (2) seconds
```

- ○ If the pick-up code is correct, a door will open, and a message "Parcel taken out" will be sent to CyberPi.
    - ▪ Parcel locker sprite:



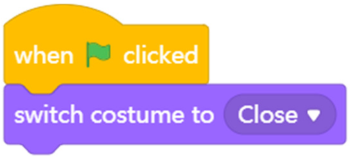
```
when I receive (Parcel taken out ▾)
switch costume to (Open ▾)
```

- ▪ CyberPi:



```
when I receive (Parcel taken out ▾)
stop (other scripts in sprite ▾)
clear screen
print (Your parcel is taken out from the locker.)
```

- ○ After taking out the parcel, click the door to close it.



```
when 🏁 clicked
switch costume to (Close ▾)
```

Section 5 – Evaluate
[5 minutes]

Objective:

- Allow students time to test their projects. Have them think about the relationship between parcel lockers and the data as well as the use of data in everyday life.

Procedures:

- Invite students to present their work.
- **Ask:** How are data used in a parcel locker system? What are the roles of data in the system?
    - **Possible answer:** The pick-up code generated by the program and the one entered by a recipient are all data. The system needs data to verify the identity of recipients.

Lesson 7

Intelligent Vehicles

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Identify the roles and importance of data in intelligent vehicles.
- Describe the working principles of autonomous lane keeping and Internet-based navigation systems in intelligent vehicles.
- Use mind maps to visualize how an intelligent vehicle project should be.
- Include functions in programs to simulate autonomous lane keeping and an Internet-based navigation system.

Key Focus

- Role of data in intelligent vehicles
- How to control intelligent vehicles

Content Standards

Type	Indicator	Standard
K12 CS Framework	Practice 5-2	Create a computational artifact for practical intent, personal expression, or to address a societal issue.
ISTE	3d	Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Preparation

For the Teacher:

- A laptop or desktop with mBlock installed
- A CyberPi kit

For Students:

- Able to write programs with coding blocks
- Prior knowledge of CyberPi’s major features

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Introduce intelligent vehicles and their applications to students. Have them understand the difference between intelligent vehicles and conventional vehicles. And encourage students to consider the roles of data in intelligent vehicles.
5 minutes	**Section 2 – Explore** • Show students what the project should be like and encourage them to consider how to achieve the effects to improve students' independent and creative thinking skills.
5 minutes	**Section 3 – Explain** • Have students use mind maps to visualize functionality analyses, preparing them for later coding session.
20 minutes	**Section 4 – Elaborate** • Give students freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Have students think about the relation between data and intelligent vehicles.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Introduce intelligent vehicles and their applications to students. Have them understand the difference between intelligent vehicles and conventional vehicles. And encourage students to consider the roles of data in intelligent vehicles.

Procedures:

- **Introduce intelligent vehicles:** Intelligent vehicles adopt intelligent driving systems, which include Internet-based navigation, autonomous driving and human intervention.
  Internet-based navigation systems help cars decide the best routes to their destinations.
  Autonomous driving technologies detect road scenes, including the status of traffic lights, road signs and more, based on which the vehicles make decisions.
  The vehicle may call for human intervention under certain conditions, requiring the driver to take over command and make decisions based on the instructions given by the intelligent system.
- **Summarize the practical application:** Countries around the world invest plenty of resources in developing intelligent vehicles. That's not only because intelligent vehicles can make people's life more convenient, but also reduces safety risks by detecting road scenes and autonomously avoiding obstacles. In addition, the development of intelligent vehicles can significantly reduce energy consumption and greenhouse gas emissions, eliminate traffic congestion, and improve social efficiency.
- **Ask:** The use of data is essential in developing intelligent driving systems. Please bear this question in mind throughout the following sessions: How do lane-keeping and navigation use data?
  Have students note down the question on paper or on the computer so that they can keep thinking about it.

Section 2 – Explore
[5 minutes]

Objective:

- Show students what the project should be like and encourage them to consider how to achieve the effects to improve students' independent and creative thinking skills.

Procedures:

- **Introduce the project:** In the picture, we see an intelligent vehicle. People could just say where to go and the vehicle will receive the oral command and start planning the best route to the destination. Without human intervention, the vehicle just automatically drives to the destination.
- **Ask:** How do we use mBlock to make an intelligent vehicle? What intelligent features should the vehicle have?
- **Summarize:** Based on your answers, the functionalities could be categorized as:
    - Lane-keeping
    - Positioning

Burger
SUPERMARKET

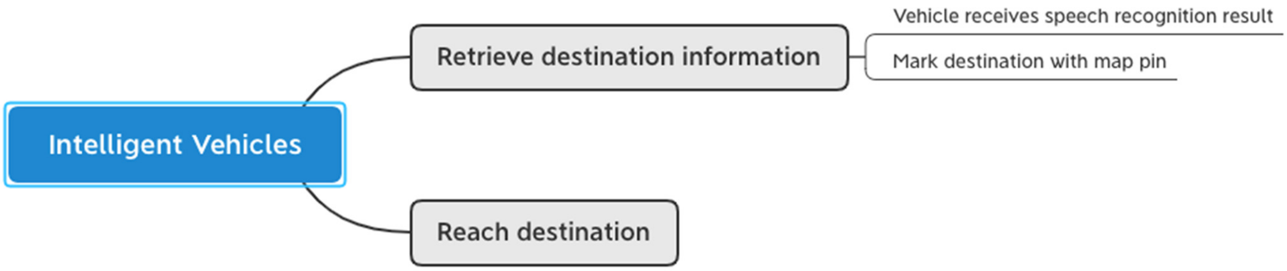
Section 3 – Explain
[5 minutes]

Objective:

- Have students use mind maps to visualize functionality analyses, preparing them for later coding session.

Procedures:

- Show students what an intelligent vehicle project should be like.
- Work with students to make a mind map to visualize the functionalities.


Intelligent Vehicles
Retrieve destination information
Vehicle receives speech recognition result
Mark destination with map pin
Reach destination

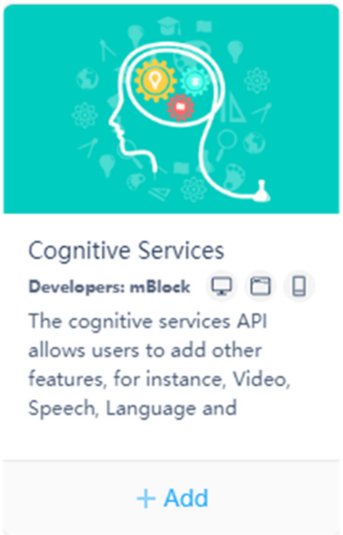
Section 4 – Elaborate
[20 minutes]

Objective:

- Give students freedom to explore further, so that they can enhance their programming skills.

Procedures:

- Add the extension **Cognitive Services** in mBlock to give your vehicle the speech recognition ability.

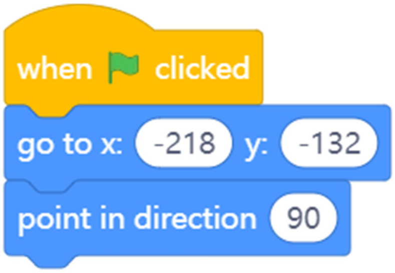


Cognitive Services

Developers: mBlock

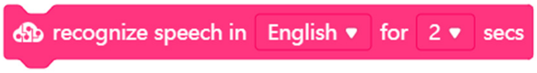
The cognitive services API allows users to add other features, for instance, Video, Speech, Language and

- Set up the starting point:

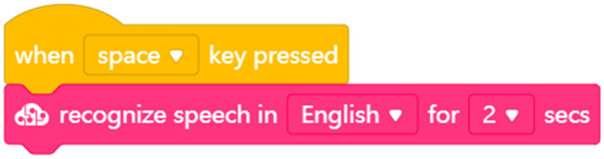


```
when 🏁 clicked
go to x: -218 y: -132
point in direction 90
```

- Retrieve destination information:
  - To recognize the oral commands, we need the speech recognition coding block below.
  - Add the Cognitive Services extension to find this block:

Coding block	Feature
 recognize speech in English ▾ for 2 ▾ secs	Use this block to recognize speech in English.

- The vehicle receives the recognition result:

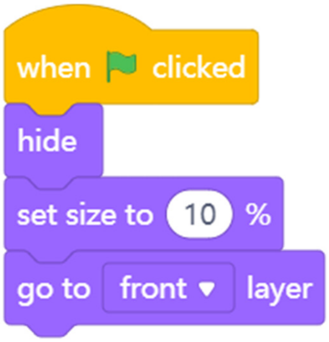


```
when space ▾ key pressed
recognize speech in English ▾ for 2 ▾ secs
```

  - *Note: The event block "when space key pressed" is used to trigger the program for detection. Students can use other events to trigger the program.*

- Mark the destination with a map pin
  - **Set up the initial status**:

    Hide the map pin before the vehicle receives the destination information.



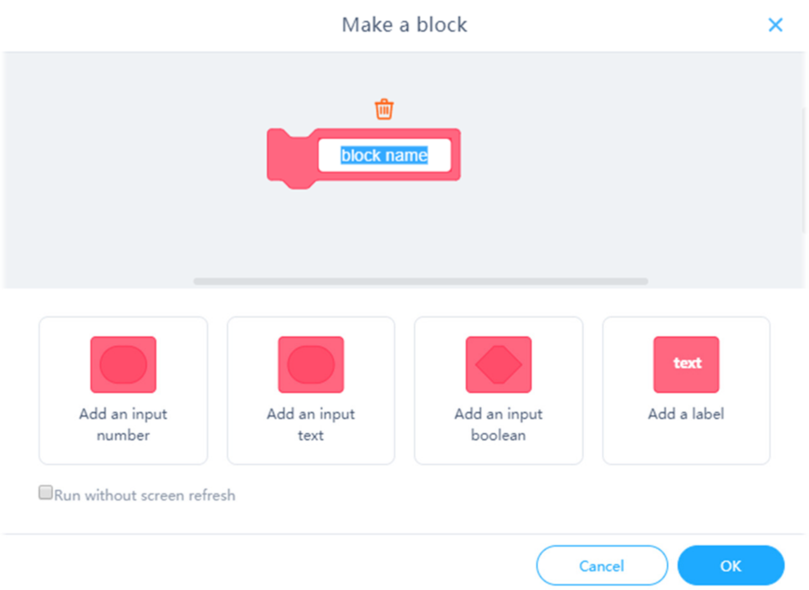
```
when 🟩 clicked
hide
set size to (10) %
go to front ▾ layer
```

  - *Note: Use "go to front layer" to locate the map pin layer at the headmost of other buildings.*

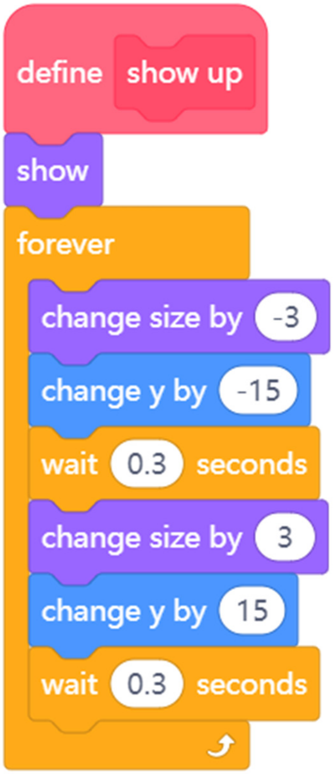
  - **Add an animated effect:**

    Program the map pin to show up with a bouncing effect, after the vehicle gets the destination information.

    - To quickly recall the code in the program, create a function named "show up" that defines a bouncing effect:

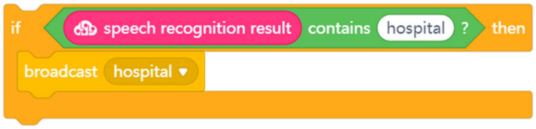
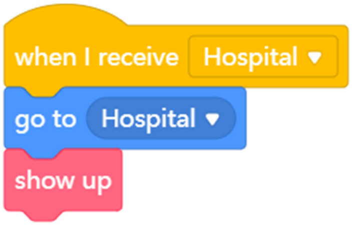
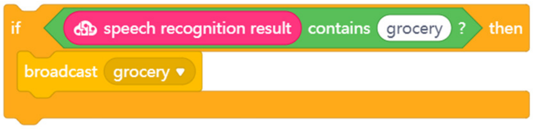
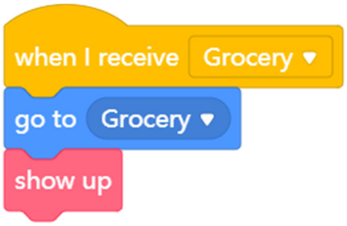


- Input "show up" to name the function. Define the function as shown:



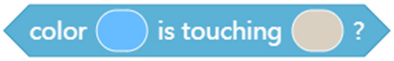
```
define show up
show
forever
    change size by -3
    change y by -15
    wait 0.3 seconds
    change size by 3
    change y by 15
    wait 0.3 seconds
```

○ **Map pin shows up at the destination after receiving the destination information:** When the vehicle receives the oral command, it sends the destination information to the map pin. Recall the function "show up" to program the map pin to appear as needed.

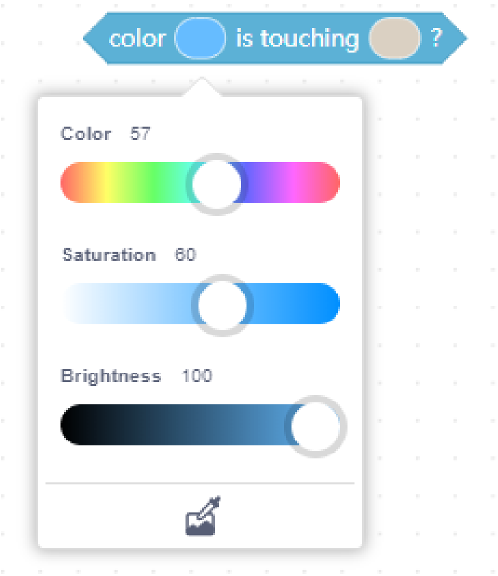
Logic	Example program (Vehicle)	Example program (Map pin)
If the destination is the hospital, **then** send a message to the map pin to have it appear around the hospital.	 if speech recognition result contains hospital ? then broadcast hospital ▾	 when I receive Hospital ▾ go to Hospital ▾ show up
If the destination is the grocery, **then** send a message to the map pin to have it appear around the grocery.	 if speech recognition result contains grocery ? then broadcast grocery ▾	 when I receive Grocery ▾ go to Grocery ▾ show up

- *Note: Use the hospital and convenient store as examples. Have students program their vehicles to get home, to a tower, to school, and other destinations.*

- The vehicle automatically drives to the destination after it receives the destination information. The vehicle should have these abilities:
  - **Autonomous lane-keeping:** The vehicle can stay in its lanes when driving
  - **Recognizing destinations:** Stops at destinations
- **Autonomous lane-keeping:**
  - To ensure the vehicle can automatically drive to destinations, program the vehicle to detect road scenesand stay in its lanes.
  - Use color-sensing blocks to help the vehicle detect road scenes so that the vehicle can automatically stay in its lane while driving. The color picker is used to precisely define the color values of the vehicle and lanes.



```
color ( ) is touching ( ) ?
```



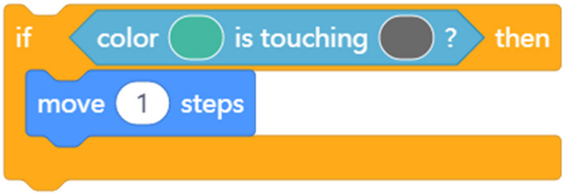
```
color ( ) is touching ( ) ?
```

Color 57

Saturation 60

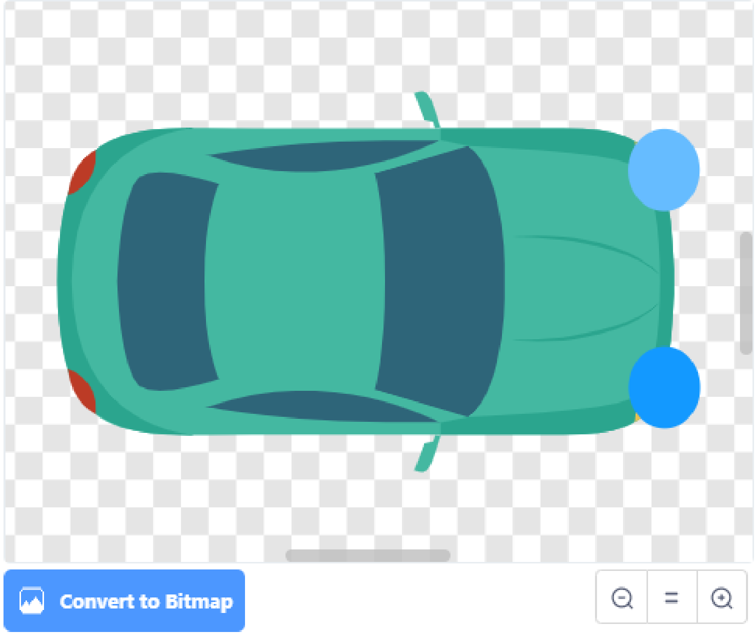
Brightness 100

- ○ **Detect roads**
  - To ensure the vehicle stays in its lanes, we use **Sensing** blocks. The vehicle is green and the lane is dark grey. Use the script below to detect whether the two colors touch each other:

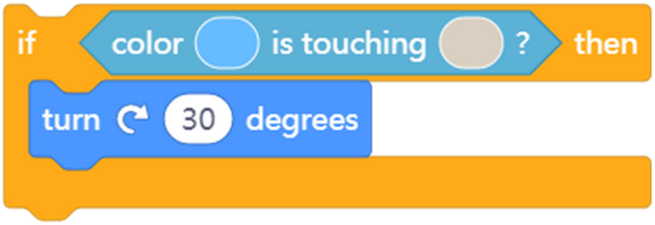


```
if < color ( ) is touching ( ) ? > then
    move (1) steps
```

- ○ **Recognize corners**
  - Program the vehicle to perform a turn when coming to a corner.
  - To detect whether to turn left or right, add a sensor to the left and one to the right of the vehicle as shown below.
  - Open the vehicle sprite's **Costumes**. Look at the two sensors at the front. The left sensor is light blue, while the right sensor is dark blue.

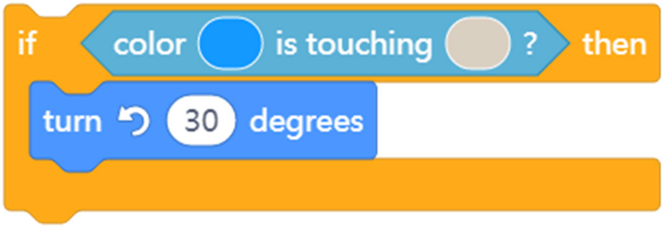


- **Decide whether to perform a turn**
  - If the left sensor touches the color outside the lane, then it's time for the vehicle to turn right.



```
if color ( ) is touching ( ) ? then
    turn ↻ 30 degrees
```

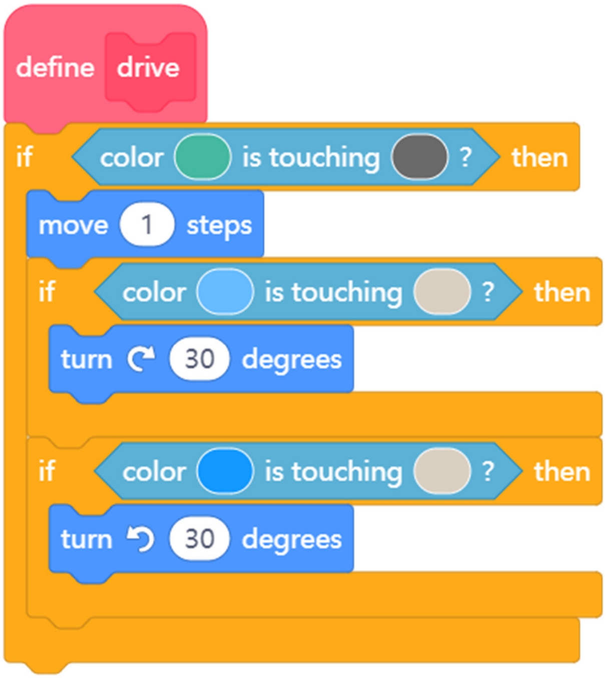
  - If the right sensor touches the color outside the lane, then it's time for the vehicle to turn left.



```
if color ( ) is touching ( ) ? then
    turn ↺ 30 degrees
```

- **Create a function named "drive"**

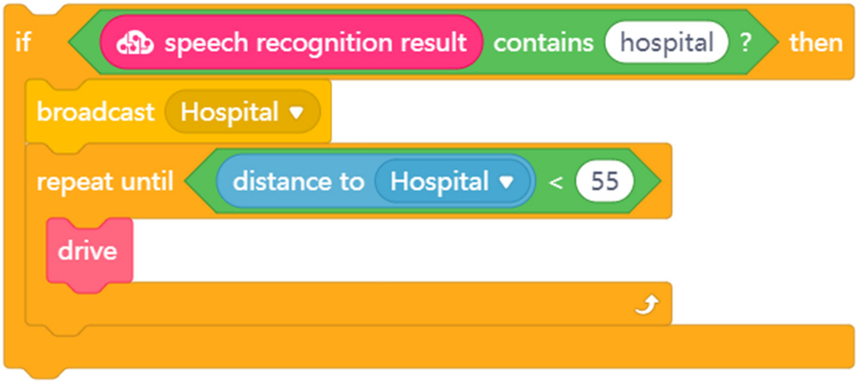
  Create a function named "drive" to include the following reusable code: Keep driving and decide whether to turn left or right.



```
define drive
if <color (●) is touching (●) ?> then
    move (1) steps
    if <color (●) is touching (●) ?> then
        turn ↻ (30) degrees
    end
    if <color (●) is touching (●) ?> then
        turn ↺ (30) degrees
    end
end
```

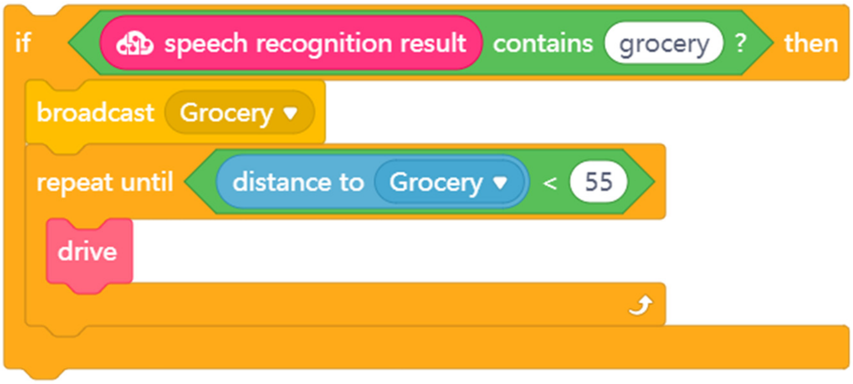
Slides 32–33

- **Recognize destinations**
    - **Destination - hospital**



```
if <(speech recognition result) contains (hospital) ?> then
    broadcast (Hospital ▾)
    repeat until <(distance to (Hospital ▾)) < (55)>
        drive
    end
end
```

- - **Destination – grocery**



```
if <(speech recognition result) contains (grocery) ?> then
    broadcast (Grocery ▾)
    repeat until <(distance to (Grocery ▾)) < (55)>
        drive
    end
end
```

- *Note:*
    - *Use hospital and grocery as examples. Have students program their vehicles to get to a tower, to school and home.*
    - *Rely on "distance" to decide whether the vehicle reaches the destinations. For instance, the car stops when the distance between it and the destination is 55. "55" is only for reference. Have students set up the distance value as appropriate.*

Section 5 – Evaluate
[5 minutes]

Objective:

- Have students think about the relation between data and intelligent vehicles.

Procedures:

- Have 1 or 2 students to share their intelligent vehicle design.
- Bring students back to the previous question, have them discuss and answer: How do you monitor lane-keeping and navigation using data? (based on the projects completed today)

Lesson 8

Bus Tracker

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Explain how a bus tracker works
- Gather real-time bus information and calculate the expected arrival time at the selected destination
- Use mind maps to analyze how to develop a bus tracker
- Program to make a bus tracker project

Key Focus

- Understanding how a bus tracker works
- Estimating bus arrival time

Content Standards

Type	Indicator	Standard
K12 CS Framework	Practice 5-2	Create a computational artifact for practical intent, personal expression, or to address a societal issue.
CSTA	2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
ISTE	3d	Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Preparation

For the Teacher:

- A desktop or laptop with mBlock installed
- A CyberPi kit

For Students:

- Basic knowledge of CyberPi
- Able to display texts, play sounds, and light up LEDs with CyberPi

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** - Describe the benefits of bus trackers with real-life examples. Introduce Intelligent Bus System and explain what roles data play in the network.
5 minutes	**Section 2 – Explore** - Describe an application scenario to students and encourage them to consider the features of the bus tracker.
5 minutes	**Section 3 – Explain** - Work with students to draw a mind map and analyze the features of the bus tracker. Prepare students for the Elaborate session.
20 minutes	**Section 4 – Elaborate** - Give students the freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** - Encourage students to present their projects and to find out more benefits of the Intelligent Bus System.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Describe the benefits of bus trackers with real-life examples. Introduce Intelligent Bus System and explain what role data plays in the network.

Procedures:

- **Ask:** Have you ever got frustrated with a long wait for a bus or when you hurried to the bus stop, only to find the bus just departed? Can you think off any solutions for this problem? So that the next time you get to the bus stop in time.
- **Give your answer:** If we have real-time bus information, we can adjust our schedule accordingly. Bus riders had to estimate arrival times in the past. Fortunately, we now can track real-time bus information with our phones.
- **Ask:** How do our phones track the bus locations and arrival times?
- **Summarize:** The technological development drives the development of an Intelligent Bus System. Travelers can view real-time bus information including bus locations, the number of passengers onboard, and traffic with the system. They can adjust their plans according to the bus location and estimated arrival time.

Section 2 – Explore
[5 minutes]

Objective:

- Describe an application scenario to students and encourage them to consider the features of a bus tracker.

Procedures:

- **Ask:** You commute between your home and school by bus every day. As a frequent traveler, what information is most important to you? And how do you get the information?
- **Summarize:** I heard some of you mentioned the real-time bus location and the arrival times at the stop near your home or school. You may also want to receive notifications when the bus is about to arrive.

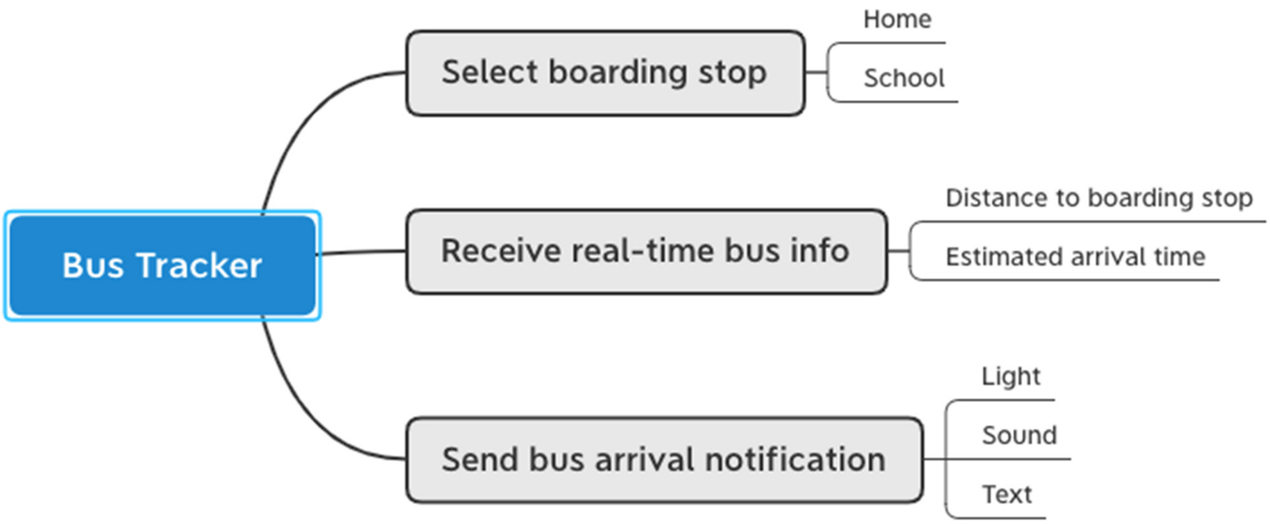
Section 3 – Explain
[5 minutes]

Objective:

- Work with students to draw a mind map and analyze the features of the bus tracker. Prepare students for the Elaborate session.

Procedures:

- Demonstrate how the bus tracker works.
- Work with students to summarize the features of a bus tracker application.
- Show students a mind map.


Bus Tracker
Select boarding stop
Home
School
Receive real-time bus info
Distance to boarding stop
Estimated arrival time
Send bus arrival notification
Light
Sound
Text

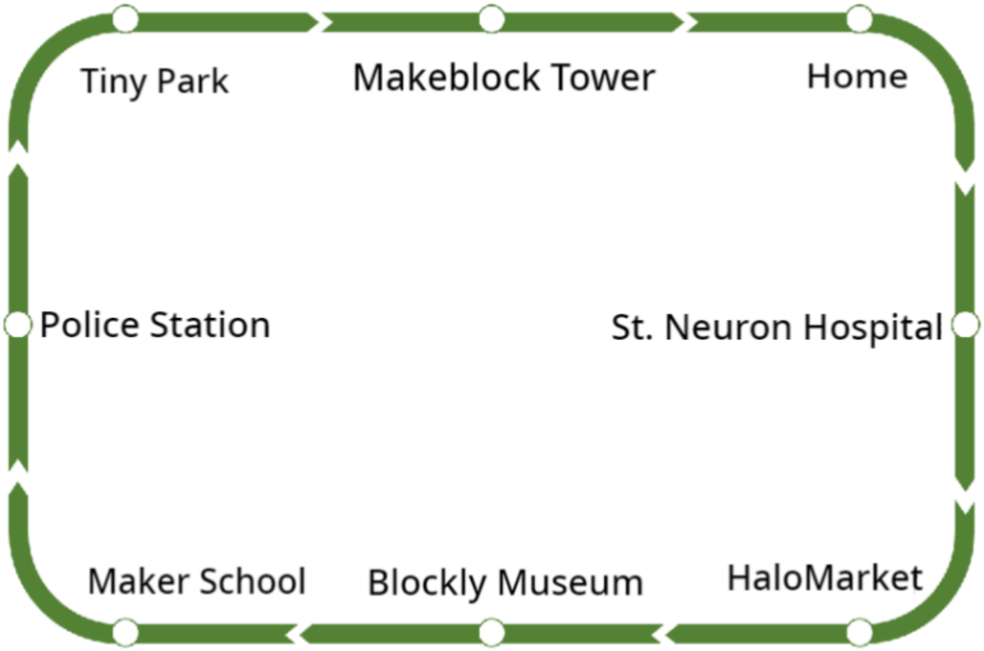
Section 4 – Elaborate
[20 minutes]

Objective:

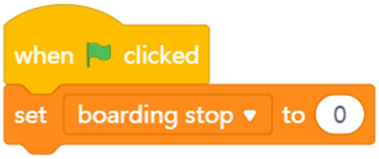
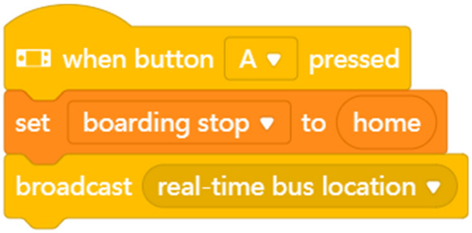
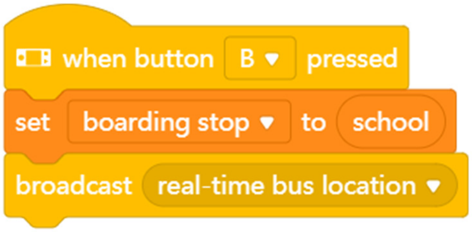
- Give students the freedom to explore further so that they can enhance their programming skills.

Procedures:

- Open the demo program. As preset, when the green flag is clicked, the bus starts moving along the circle line.
- Connect CyberPi to mBlock and select the **Live** mode.
- Select a boarding stop:
  - The circle line starts at the Tiny Park and travels clockwise. The **Tiny Park** is **Stop 1**, **Home Stop 3**, and **Maker School Stop 7**.



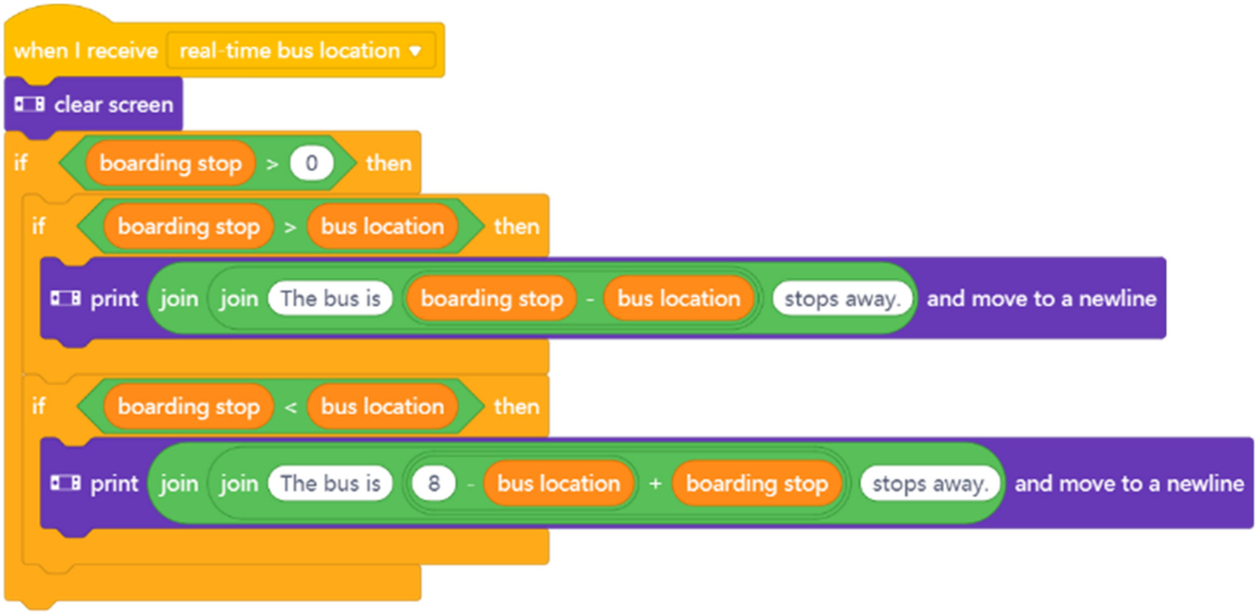
- ○ Create a variable "boarding stop" to record the number of the stop where the passenger is currently at.

Logic	Example program (CyberPi)
When program starts up Set boarding stop to 0	 when 🏁 clicked set boarding stop ▾ to 0
When button A pressed Select home as boarding stop Receive real-time bus location	 when button A ▾ pressed set boarding stop ▾ to home broadcast real-time bus location ▾
When button B pressed Select school as boarding stop Receive real-time bus location	 when button B ▾ pressed set boarding stop ▾ to school broadcast real-time bus location ▾

- ○ *NOTE:*
  - *When CyberPi starts up, no boarding stop is selected yet. So, set the variable "boarding stop" to 0.*
  - *In the preset program, the variables "home" and "school" are set to 3 and 7 respectively.*

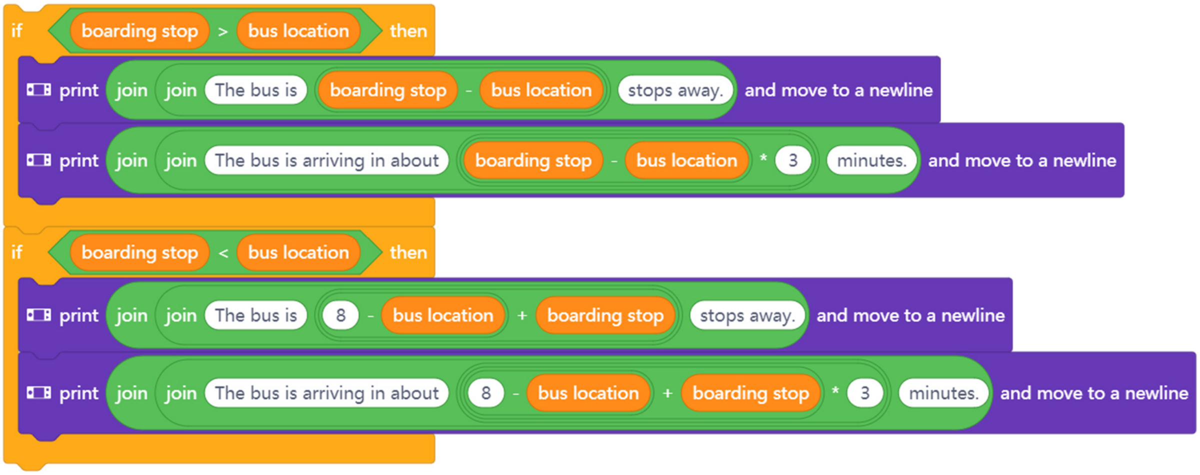
- **Receive real-time bus information**
- **Distance to boarding stop**
  - ○ In the preset program, a variable "bus location" is created to record the stop where the bus is currently at.
  - ○ The numbers 1 to 8 represent the eight stops on the circle line.

- Three relationships between "boarding stop" and "bus location":
    - boarding stop > bus location—Bus still not coming
    - boarding stop < bus location—Bus already left. Wait for the next one
    - boarding stop = bus location—Bus arrives
- **Prompt notification:**



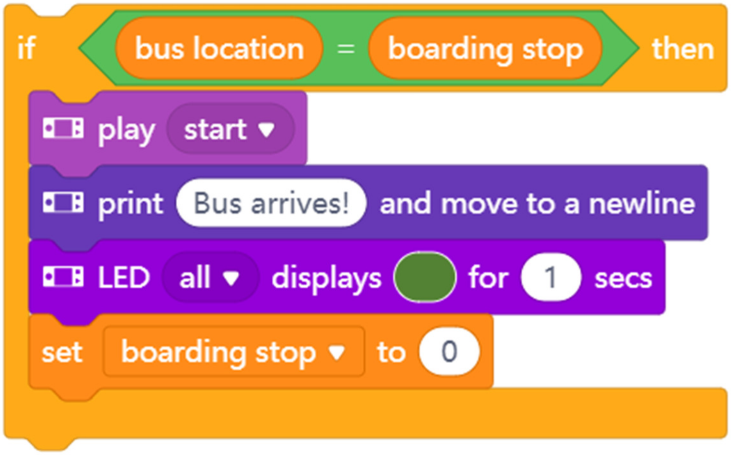
```
when I receive [real-time bus location ▾]
clear screen
if <(boarding stop) > (0)> then
    if <(boarding stop) > (bus location)> then
        print (join (join (The bus is) ((boarding stop) - (bus location))) (stops away.)) and move to a newline
    if <(boarding stop) < (bus location)> then
        print (join (join (The bus is) (((8) - (bus location)) + (boarding stop))) (stops away.)) and move to a newline
```

- **Estimated arrival time:**
    - It takes three minutes for the bus to move from a stop to the next.
    - Estimated arrival time = Distance to boarding stop × 3. For instance, if the bus is two stops away from the boarding stop, the estimated arrival time will be six minutes.
    - **Display estimated arrival time:**



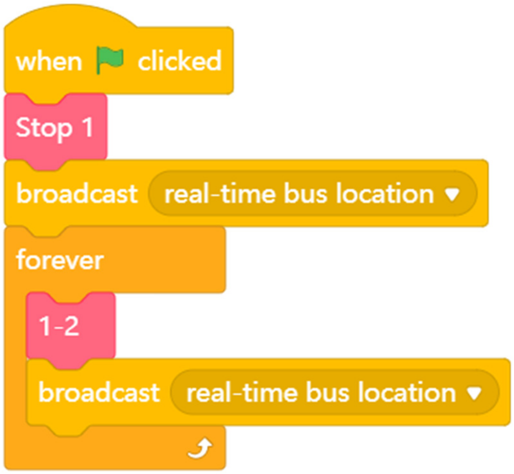
```
if <(boarding stop) > (bus location)> then
    print (join (join (The bus is) ((boarding stop) - (bus location))) (stops away.)) and move to a newline
    print (join (join (The bus is arriving in about) (((boarding stop) - (bus location)) * (3))) (minutes.)) and move to a newline
if <(boarding stop) < (bus location)> then
    print (join (join (The bus is) (((8) - (bus location)) + (boarding stop))) (stops away.)) and move to a newline
    print (join (join (The bus is arriving in about) ((((8) - (bus location)) + (boarding stop)) * (3))) (minutes.)) and move to a newline
```

- **Send bus arrival notification**



```
if <(bus location) = (boarding stop)> then
    play (start ▾)
    print (Bus arrives!) and move to a newline
    LED (all ▾) displays (🟩) for (1) secs
    set (boarding stop ▾) to (0)
```

- **Automatic updates on bus information:**
  - After a passenger sets the location, the bus information will be automatically updated every time the bus arrives at a stop. The updates stop when the bus reaches the stop where the passenger is at.
  - Select the sprite "Bus" and view the preset programs for the sprite. The custom block "1-2" means the bus travels from Stop 1 to Stop 2. Other custom blocks, like "2-3", comply with the same logic.
  - To receive real-time bus information, add a **broadcast ( )** block after the initial stop and each custom block. Name the broadcast message as "real-time bus location".



```
when 🏁 clicked
Stop 1
broadcast (real-time bus location ▾)
forever
    1-2
    broadcast (real-time bus location ▾)
↻
```

Section 5 – Evaluate
[5 minutes]

Objective:

- Encourage students to present their projects and to find out more benefits of the Intelligent Bus System.

Procedures:

- Invite one to two volunteers to present their bus trackers to the class.
- Ask students to discuss this question: In what aspects can you improve the local bus system?
- Have one to three students present their conclusion.

Lesson 9

Colors Hunter

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Describe basic game design principles and explain the mechanics of a game
- Use mind maps to analyze what features the game Colors Hunter should have
- Use variables to make the game – Colors Hunter

Key Focus

- Understanding some basic game design principles, and analyzing the mechanics of a game
- Using mind maps to visualize features the game should have, and program the game based on the mind map

Content Standards

Type	Indicator	Standard
K12 CS Framework	Practice 3-1	Create a computational artifact for practical intent, personal expression, or to address a societal issue.
CSTA	2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.

Preparation

For the Teacher:

- A desktop or laptop with mBlock installed
- A CyberPi kit

For Students:

- Basic knowledge of how sensors work
- With experience in block-based coding, and able to apply variables

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Have students figure out what phases a game often has, and let them categorize these phases to introduce them to the game design process.
5 minutes	**Section 2 – Explore** • Show students how the game works, and encourage them to consider how to turn their ideas into reality so that they can develop their independent thinking skills.
5 minutes	**Section 3 – Explain** • Work with students to analyze the game mechanics using a mind map, preparing them for the later programming activities.
20 minutes	**Section 4 – Elaborate** • Give students freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Ask students to try the game they just created, and have them continue to practice game design.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Have students figure out what stages a game often has, and let them categorize these stages to engage them in the game design process.

Procedures:

- Ask students to think about how many phases most games often have.
- **Summarize the answers:**

  Considering the gaming experience, we could divide a game into 3 phases: Opening, Middlegame, Endgame.
- **Say:** Suppose you are a game designer, so your responsibility is to design an engaging game. You have many things to consider when you're designing a game:

  (1) Task: What's the task that a player needs to complete?

  (2) Controls: How does a player interact with the game?

  (3) Outcome: What outcome will the player get? How is this outcome reached?

- Use specific game examples for analysis.
- **Ask:** Pick a game that most people are familiar with. Analyze its tasks, controls and outcome.

Section 2 – Explore
[5 minutes]

Objective:

- Show students how the game works and encourage them to consider how to turn their ideas into reality. To help them develop their independent thinking skills.

Procedures:

- Invite students to try the Colors Hunter game.
- **Ask:** If you are a game designer, what do you think the task and control methods are? And what's the outcome?
- **Summarize:**

  Task: Follow CyberPi's command to select a light color accordingly.

  Control method: Use the joystick to select a color.

  Outcome: When the 30-second countdown stops, the game is over. There's no win or lose condition.

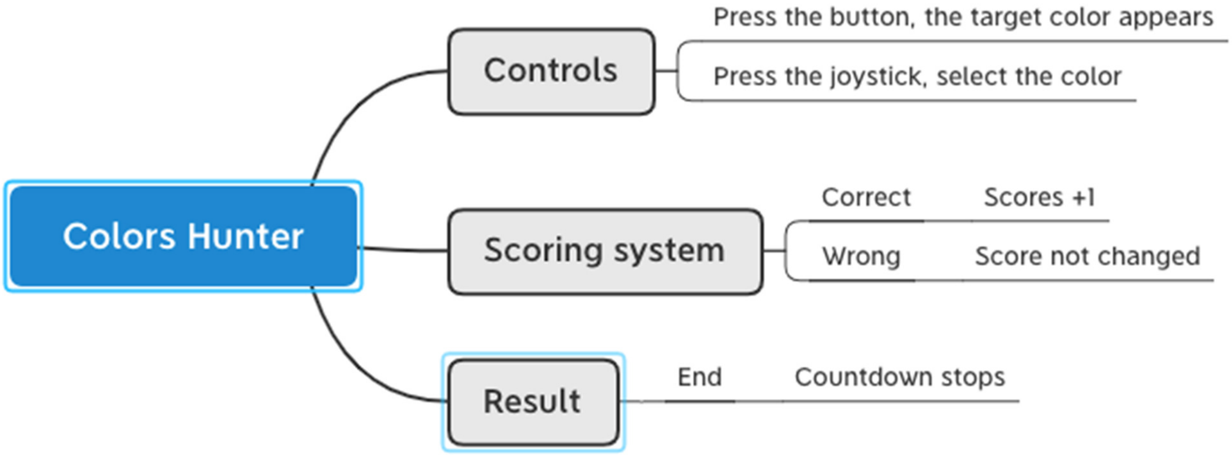
Section 3 – Explain
[5 minutes]

Objective:

- Work with students to analyze the game mechanics using a mind map, preparing them for the later programming activities.

Procedures:

- Work with students to analyze what features the game must have.
- Show students the mind map.



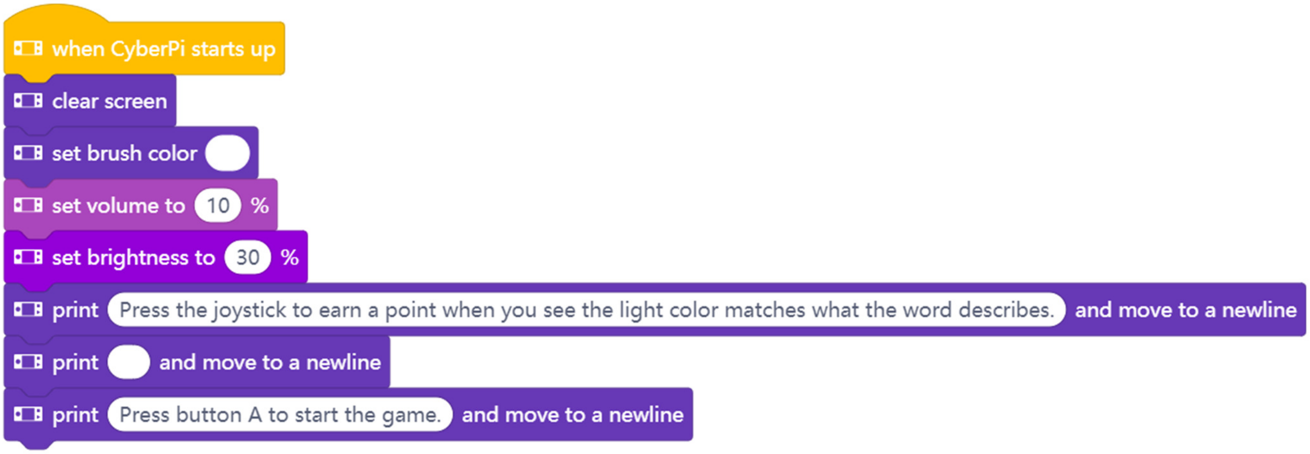
Section 4 – Elaborate
[20 minutes]

Objective:

- Give students freedom to explore further so that they can enhance their programming skills.

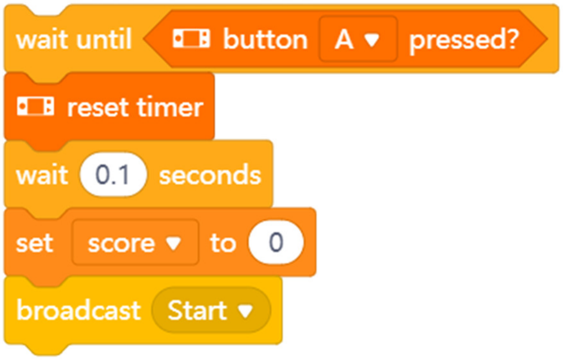
Procedures:

- Connect CyberPi to mBlock.
- Press the button to start the game:
    - Add a prompt to indicate the start of the game.



```
when CyberPi starts up
clear screen
set brush color ( )
set volume to (10) %
set brightness to (30) %
print (Press the joystick to earn a point when you see the light color matches what the word describes.) and move to a newline
print ( ) and move to a newline
print (Press button A to start the game.) and move to a newline
```

- Press the button, and the game starts:



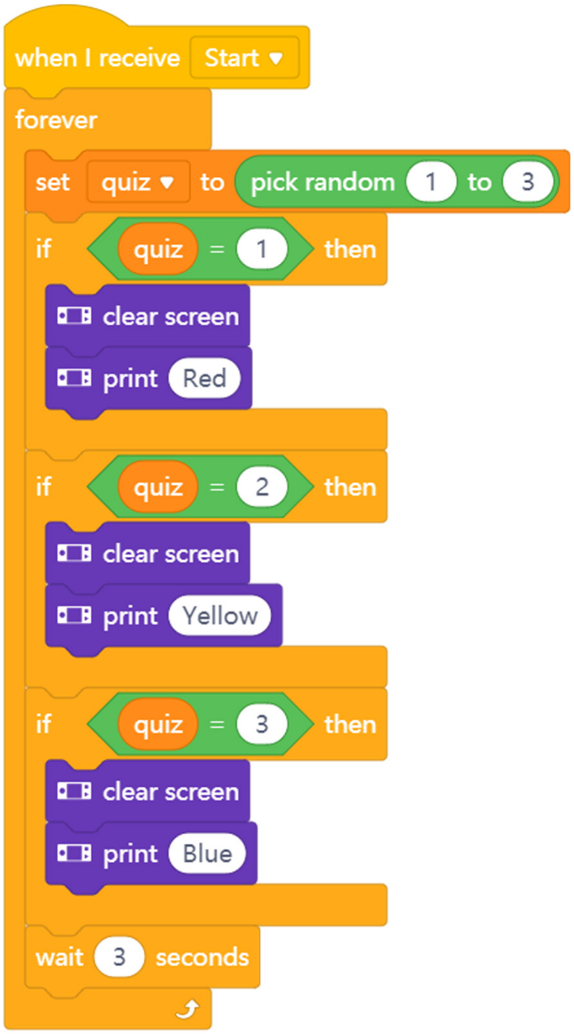
```
wait until <button (A ▼) pressed?>
reset timer
wait (0.1) seconds
set (score ▼) to (0)
broadcast (Start ▼)
```

- *Note:*

  ① The variable "score" is created to store the scores. Make sure the scores are cleared before the game begins.

② The timer tracks the gaming time, and should be reset to zero at the very beginning of the game.

- The system randomly selects a target:
    - CyberPi will randomly display one of the three words, "red", "yellow" and "blue", on its screen each single time.
    - Use a variable "target" in the program to simulate a test in which a target color is randomly given each time.
    - For example, when "target" equals 1, the word "red" appears on the screen; when "target" equals 2, the word "yellow" appears on the screen; when "target" equals 3, the word "blue" appears on the screen.



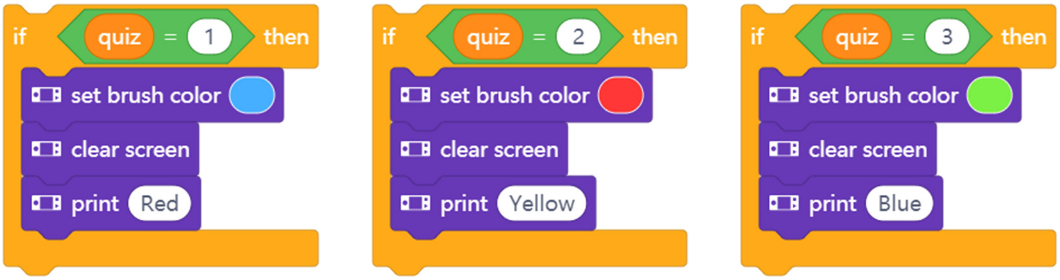
```
when I receive Start
forever
    set quiz to pick random 1 to 3
    if quiz = 1 then
        clear screen
        print Red
    if quiz = 2 then
        clear screen
        print Yellow
    if quiz = 3 then
        clear screen
        print Blue
    wait 3 seconds
```

- *Note:*

① Use the "forever" block to keep refreshing the proposed targets;

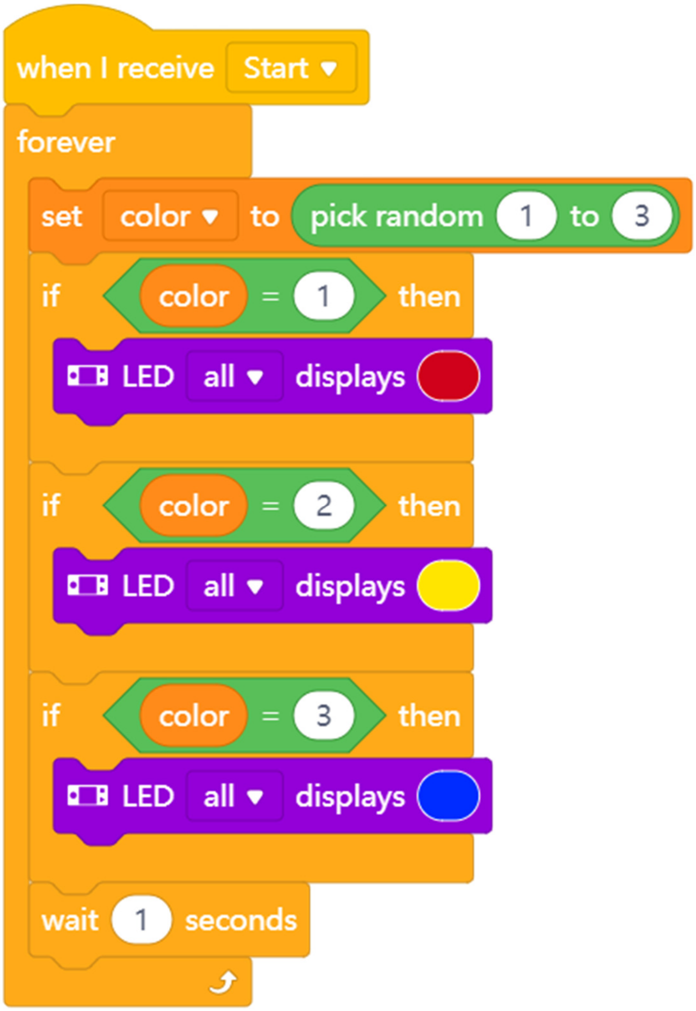
② Change the value in the "wait () secs" to adjust the level of difficulty.

- Set up changes:
  - To make the game more challenging and engaging, it's better to add some changes.
  - For example, program the texts to appear in random colors on the screen.



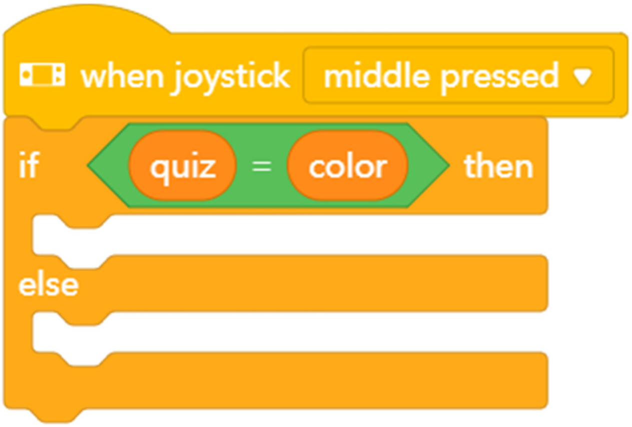
```
if quiz = 1 then
    set brush color 🟦
    clear screen
    print Red

if quiz = 2 then
    set brush color 🟥
    clear screen
    print Yellow

if quiz = 3 then
    set brush color 🟩
    clear screen
    print Blue
```

- Press the joystick to confirm the color. After CyberPi gives a target color, the player is expected to pinpoint the target among changing colors using the joystick.
  - Keep colors changing:
    - Create a variable named "LightColor" to change the colors of the lights. The variable is used to decide whether the player's selection is correct as well.



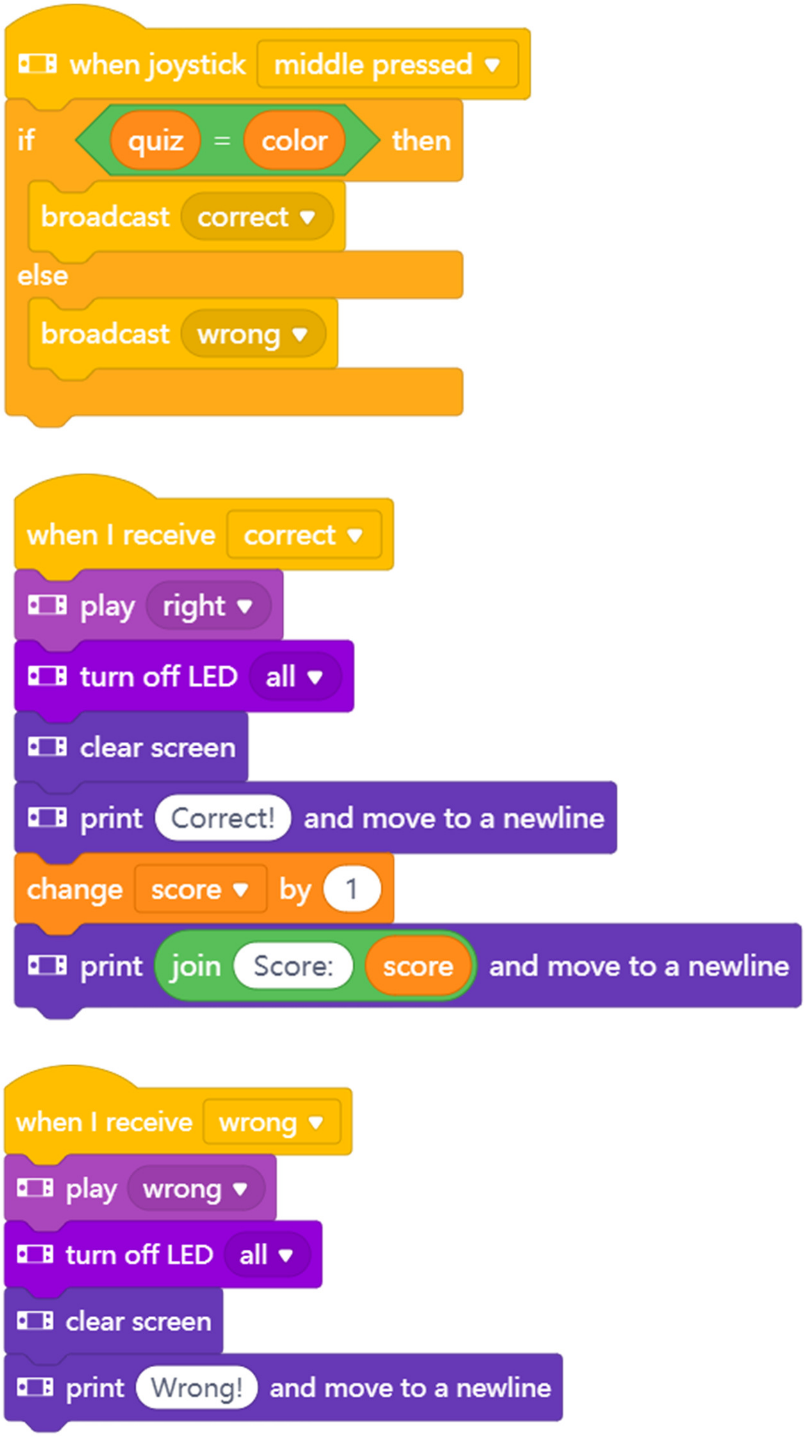
```
when I receive Start ▾
forever
    set color ▾ to pick random 1 to 3
    if color = 1 then
        LED all ▾ displays 🔴
    if color = 2 then
        LED all ▾ displays 🟡
    if color = 3 then
        LED all ▾ displays 🔵
    wait 1 seconds
```

    - *Note: Use the block "wait (1) sec" to allow the player time to complete his or her selection. The player faces greater challenges when the interval time is shorter.*
- Press the joystick to confirm color:
    - Compare the target and the current light color



```
when joystick middle pressed ▾
if quiz = color then

else

```

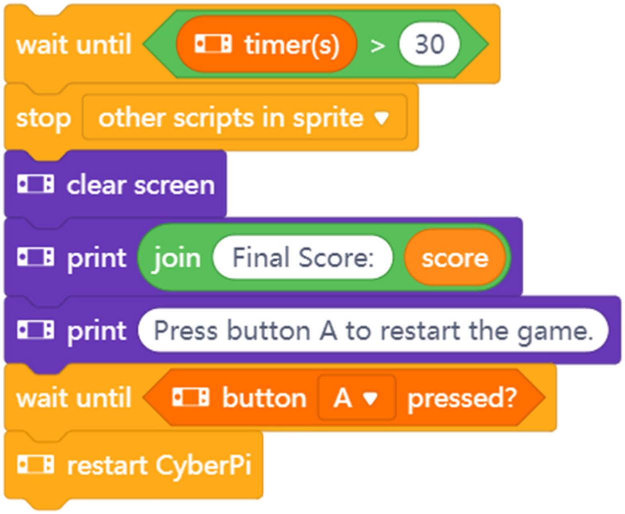
- Scoring system: Create a variable "score" to store the points that the player wins. The score increases by 1 when the player's selection matches the target.



```
when joystick [middle pressed ▾]
if <(quiz) = (color)> then
    broadcast [correct ▾]
else
    broadcast [wrong ▾]

when I receive [correct ▾]
play [right ▾]
turn off LED [all ▾]
clear screen
print (Correct!) and move to a newline
change [score ▾] by (1)
print (join (Score:) (score)) and move to a newline

when I receive [wrong ▾]
play [wrong ▾]
turn off LED [all ▾]
clear screen
print (Wrong!) and move to a newline
```

  - *Note: Use sounds and texts to let the player know whether he or she picks the correct color.*

- Result: The countdown starts when the game begins. After 30 seconds, the game stops and shows the final score.



```
wait until < timer(s) > 30 >
stop other scripts in sprite ▾
clear screen
print join Final Score: score
print Press button A to restart the game.
wait until < button A ▾ pressed? >
restart CyberPi
```

Section 5 – Evaluate
[5 minutes]

Objective:

- Ask students to try the game they just created and have them continue to practice game design.

Procedures:

- Have 2 students compete to see who gets the highest scores
- Have students create a simple game based on the principles mentioned today:

  task, controls, and outcome.

Lesson 12

Opposite Game

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Explain the roles of data in designing game mechanics and develop a simple game
- Use variables and lists to establish rules for an opposite game and write programs to decide the game results
- Draw mind maps to analyze how to develop an opposite game
- Improve reaction time and reverse thinking skills by designing and developing the game

Key Focus

- Using variables and lists to establish rules for an opposite game and deciding the game results

Content Standards

Type	Indicator	Standard
CSTA	2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.

Preparation

For the Teacher:

- A desktop or laptop with mBlock installed
- A CyberPi kit

For Students:

- Understanding on how to display images and text on CyberPi's screen
- Using variables and lists

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Engage students in a fun warmup activity, the Opposite Game. Encourage them to think about how the decision system of the game works.
5 minutes	**Section 2 – Explore** • Have students brainstorm how a game decides whether a player wins.
5 minutes	**Section 3 – Explain** • Work with students to draw a mind map and analyze the game mechanics, preparing them for the Elaborate session.
20 minutes	**Section 4 – Elaborate** • Give students the freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Improve students' engagement and encourage them to complete the project by challenging them to the opposite game.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Engage students in a fun warmup activity, the Opposite Game. Encourage them to think about how the decision system of the game works.

Procedures:

- **Warmup activity—Opposite Game**
  Ask students to do the opposite of your commands. For example, when you say "look up", students should look down; when you say "lift your right hand", they should lift their left hand. Tell students the correct action after they perform an action.

- **Ask:** The opposite game can improve reaction time and concentration. Now, I want you to recap how we decided whether an action was correct.
- **Summarize:** In the opposite game, every command has a designated action, like stand up and sit down. If you do the opposite of the command, then you do the correct action. Many games adopt a similar method to decide the results, comparing players' actions with the preset designated actions. If a player's actions match the actions set in the system, then the player wins the game.

Section 2 – Explore
[5 minutes]

Objective:

- Have students brainstorm how a game decides whether a player wins.

Procedures:

- Introduce the task: We're going to use the same decision mechanism to develop a simple opposite game.
- **Ask:** How do we decide whether a player wins or loses the game?
- **Summarize:** When a command comes up, the player needs to perform an action. If the player does the opposite of the command within the given time, then the player wins the game. If the player fails to do the opposite or the time is up, then the player loses the game.

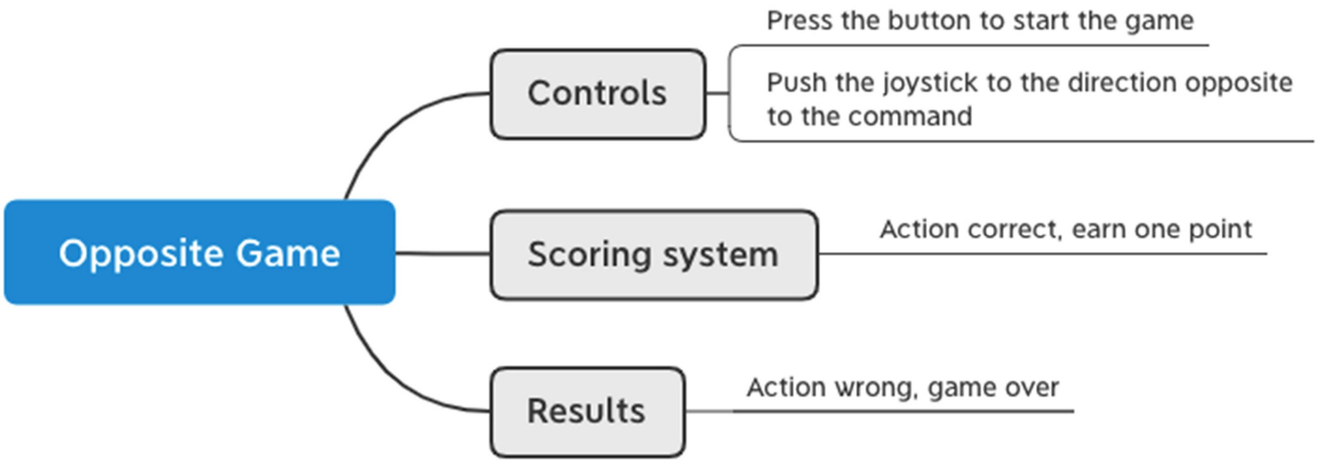
Section 3 – Explain
[5 minutes]

Objective:

- Work with students to draw a mind map and analyze the game mechanics, preparing students for the Elaborate session.

Procedures:

- Work with students to analyze the game mechanics.
- Show the mind map to students.


Opposite Game
Controls
Press the button to start the game
Push the joystick to the direction opposite to the command
Scoring system
Action correct, earn one point
Results
Action wrong, game over

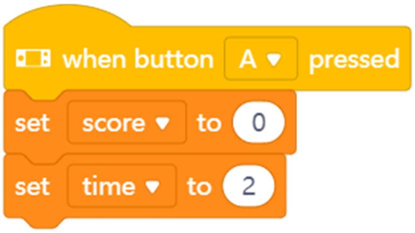
Section 4 – Elaborate
[20 minutes]

Objective:

- Give students the freedom to explore further so that they can enhance their programming skills.

Procedures:

- Open mBlock, add CyberPi in **Devices**, connect CyberPi to mBlock, and select **Live** mode.
- To make it easier to check whether players' actions match the designated ones, create a list "command" to store the commands.
    - Add four commands to the list, "up", "down", "left", and "right".
- Press the button to start a game:
    - The game begins:



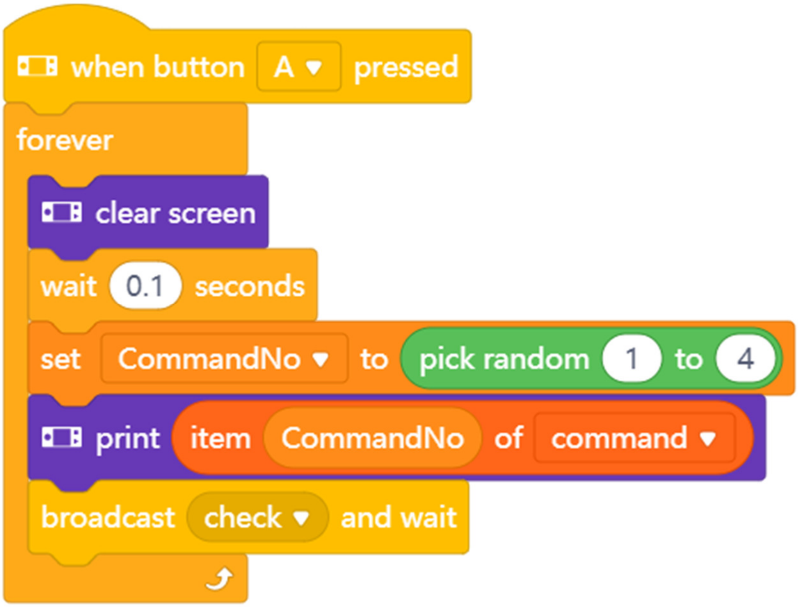
```
when button A pressed
set score to 0
set time to 2
```

- *NOTE:*

  ① Use the variable "score" to record a player's scores.

  ② Use the variable "time" to set the time limit. Set "time" to 2. If players fail to do an action in 2 seconds, they lose the game.

    - Display commands: The game starts when the button is pressed. CyberPi should display a command as the game starts.



```
when button A ▾ pressed
forever
    clear screen
    wait 0.1 seconds
    set CommandNo ▾ to pick random 1 to 4
    print item CommandNo of command ▾
    broadcast check ▾ and wait
```

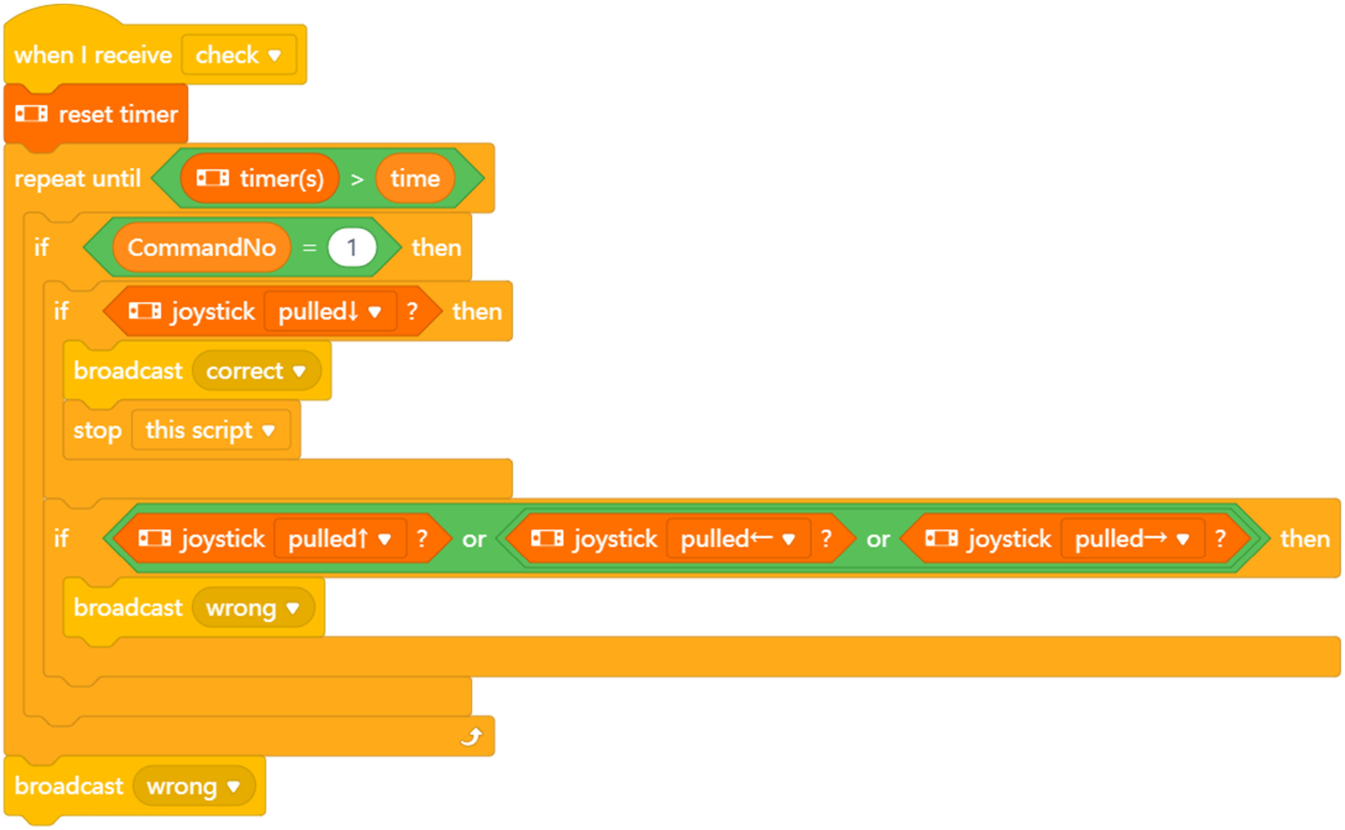
- *NOTE:*

  ① To make sure players are clear about which command to follow, clear the screen every time before displaying the next command.

  ② Use the variable "CommandNo" to match the number of each item in the "commands" list. As the list includes four commands, pick a random number between 1 and 4 to display the commands randomly.

  *③ Include a **broadcast ( ) and wait** block in this script to report the decision about whether the player makes the right action before the next command comes up.*

- Push the joystick to the direction opposite to the command:
  - If the player does the opposite of the command within the given time, then the player wins the game.
  - If the player does the same as the command or if the time is up, then the player loses the game.



```
when I receive [check ▾]
reset timer
repeat until < (timer(s)) > (time) >
    if < (CommandNo) = (1) > then
        if < joystick [pulled↓ ▾] ? > then
            broadcast [correct ▾]
            stop [this script ▾]
        if < < joystick [pulled↑ ▾] ? > or < < joystick [pulled← ▾] ? > or < joystick [pulled→ ▾] ? > > > then
            broadcast [wrong ▾]
broadcast [wrong ▾]
```

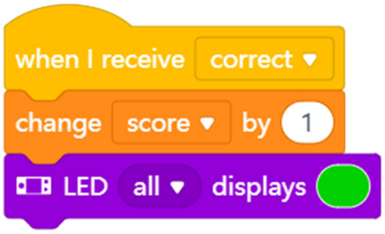
- *NOTE:*

  ① The example program includes the decision system for the "up" command. Complete the programs for the rest commands. When "CommandNo" equals 1, the command is "up". If the player manages to push down the joystick within the given time, then the player makes a correct action and wins the game. If the player pushes the joystick to other directions or if the time is up, the player loses the game.

  ② If the player's action is correct, stop the script and display another command. If the player's action is wrong, the game is over.

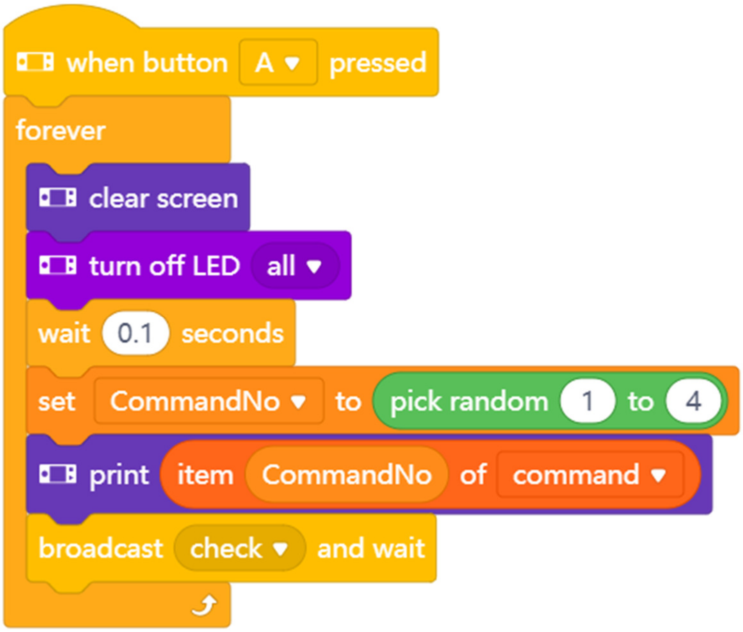
  ③ To check whether a player performs the action within the given time, compare the variable "time" with the timer. When the timer value is greater than the maximum value "time", it means the player exceeds the maximum time, and the decision system will stop working.

- Scoring system
  - Action correct, earn one point: When the player's action is correct, "score" increases by 1, and CyberPi's LEDs light up green.



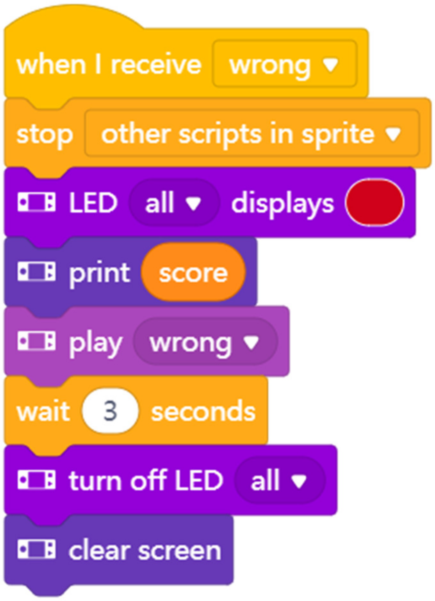
```
when I receive [correct ▾]
change [score ▾] by (1)
LED [all ▾] displays 🟩
```

- ○ Display next command and turn off LEDs: When the player's action is correct, CyberPi displays the next command. The screen should be cleared and the LEDs turned off before CyberPi displays the next command.



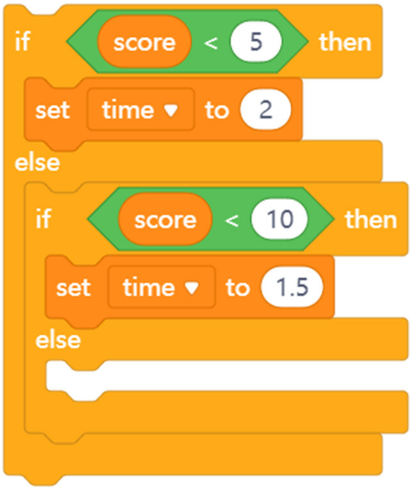
```
when button [A ▾] pressed
forever
    clear screen
    turn off LED [all ▾]
    wait (0.1) seconds
    set [CommandNo ▾] to (pick random (1) to (4))
    print (item (CommandNo) of [command ▾])
    broadcast [check ▾] and wait
```

- Action wrong, game over: When the player's action is wrong, the game ends. CyberPi will display the score, and its LEDs will light up red.



```
when I receive wrong ▾
stop other scripts in sprite ▾
LED all ▾ displays ●
print score
play wrong ▾
wait 3 seconds
turn off LED all ▾
clear screen
```

- Results: When the player does a wrong action, the game ends.
  - Make the game more challenging and more interesting.
  - For example, set up the scoring system based on the levels of difficulty. In the harder version, players have less time to react but earn more points for a correct action.



```
if score < 5 then
    set time ▾ to 2
else
    if score < 10 then
        set time ▾ to 1.5
    else

```

- *NOTE:*

  ① The example program provides two levels of difficulty. Build on the example program to diversify the levels of difficulty.

Points	Time	Points	Time
0~4	2	15~19	0.5
5~9	1.5	20~24	0.3
10~14	1	25~∞	0.2

② *Run the script above as soon as the game begins.*

Section 5 – Evaluate
[5 minutes]

Objective:

- Improve students' engagement and encourage them to complete the project by challenging them to the opposite game.

Procedures:

- **Activity 1:** Divide students into groups and have them play the game to see which group finishes the most rounds.
- **Activity 2:** Invite five to six students to compete in front of the class.
  - **Tip:** Pick one of the activities according to class performance. To ensure fair competition, invite other students to examine the players' programs.

Lesson 11

Catch Fish Carnival Game I

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Identify the role of data in showing game results, and the importance of scores and rankings in a game.
- Clone sprites and program the clones to perform tasks.
- Calculate the score by rounding the quotient to an integer or reporting the remainder.

Key Focus

- Understanding of the role of data in showing game results, and the importance of scores and rankings in a game.
- Cloning sprites and programming clones to perform tasks.
- Performing calculations on the score by rounding the score to an integer or reporting the remainder.

Content Standards

Type	Indicator	Standard
CSTA	2-AP-11	Create clearly named variable that represent different data types and perform operations on their values.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
CSTA	2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
ISTE	5c	Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

Preparation

For the Teacher:

- A desktop or laptop with mBlock installed
- A CyberPi kit

For Students:

- Practical knowledge on how to change the sprites' costumes and movements, and use broadcasts.
- Understanding of the basic principles of game design.

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Have students consider the importance of scores and rankings in a game. Use some well-known games as examples, helping students understand the role of data in showing the results of a game.
5 minutes	**Section 2 – Explore** • To help students build an overall picture of the project, ask them to discuss and summarize the game rules after they try the game.
5 minutes	**Section 3 – Explain** • Have students analyze the mechanics of the game through a mind map, preparing them for the later coding activities.
20 minutes	**Section 4 – Elaborate** • Give students freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Have students test their games and compete with each to see who wins the highest score.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Have students consider the importance of scores and rankings in a game. Use some well-known games as examples, helping students understand the role of data in showing the results of a game.

Procedures:

- **Ask:** Players often see the result at the end of a game. The result could be a score, stars, etc. Imagine how the player will react if he or she can't see any results at the end of a game.
- **Summarize:** Without the results, the player may be confused about how they did, so the gaming experience could be poor. A good game gives the player a sense of accomplishment, and it leaves much room for the player to pursue higher scores and rankings and challenge more difficult levels.

Section 2 – Explore
[5 minutes]

Objective:

- Introduce the project to students so that they can have an overall picture of it and ask them to summarize the game rules after a discussion with partners.

Procedures:

- Invite 1 or 2 students to try the Catch Fish Carnival game.
- Ask students to have a discussion:

  1) What's the goal of the game?

  2) How does a player earn points?

  3) How is the result shown?
- Summarize: Press the **Start** button to begin. The score increases by 1 each time you catch a fish. You have 30 seconds and when the time is up, the final score will be shown.

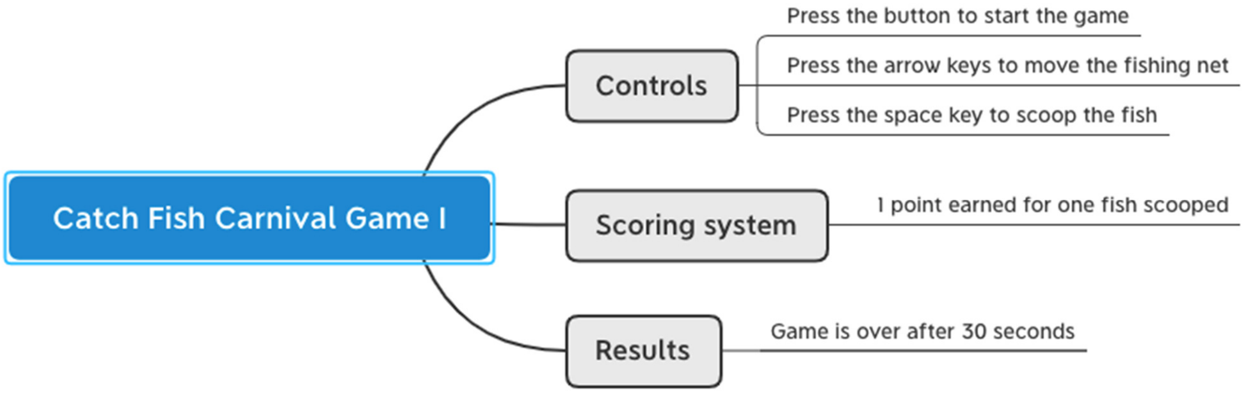
Section 3 – Explain
[5 minutes]

Objective:

- Have students analyze the mechanics of the game through a mind map, preparing them for the later coding activities.

Procedures:

- Work with students to summarize the main functions of the project.
- Work with students to create a mind map to clarify what features the project needs.


Catch Fish Carnival Game I
Controls
Press the button to start the game
Press the arrow keys to move the fishing net
Press the space key to scoop the fish
Scoring system
1 point earned for one fish scooped
Results
Game is over after 30 seconds

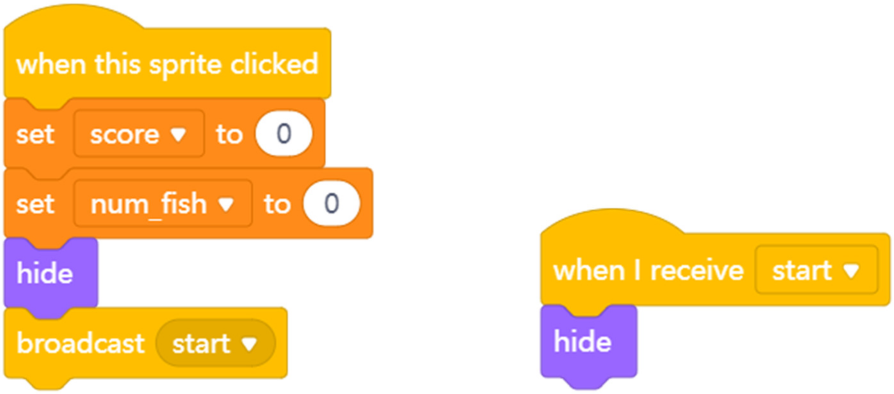
Section 4 – Elaborate
[20 minutes]

Objective:

- Give students freedom to explore further so that they can enhance their programming skills.

Procedures:

- Ask students to open the example program file. As defined by the preset program, when the green flag is clicked, the start screen appears.
- Press the **Start Game** button to begin:
    - The game starts:



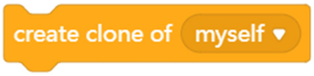
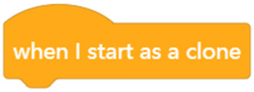
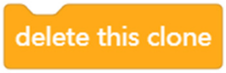
```
when this sprite clicked
set score ▾ to 0
set num_fish ▾ to 0
hide
broadcast start ▾
```

```
when I receive start ▾
hide
```

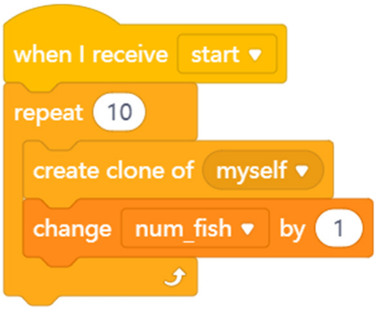
        - *NOTE:*

          ① Create a variable "score" to record game scores.

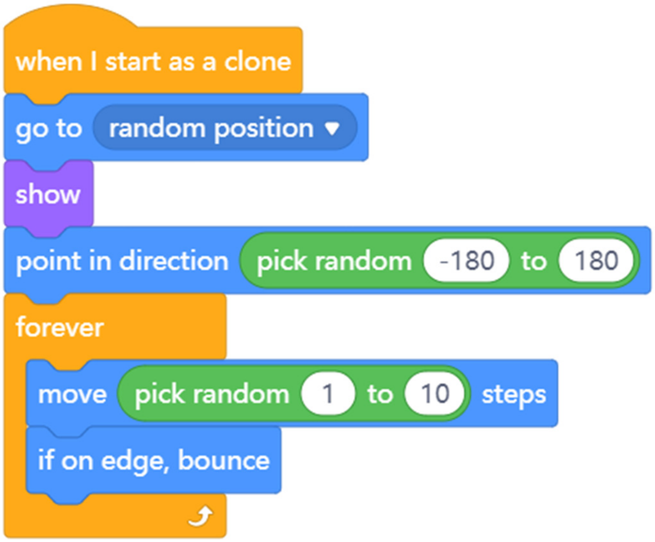
          ② Create a variable "num_fish" to record the number of fish on the screen.
    - Multiple fish randomly swimming:
        - Introduce cloning: To create multiple fish swimming on the screen, we need to clone sprites. Simply speaking, cloning is to duplicate multiple identical sprites. Cloning helps reduce repetitive coding while at the same time creating identical sprites.

Coding block	Feature
 create clone of myself	Use this block to create a clone of a specified sprite. The clone generated completely overlaps the original one. Select a sprite from the drop-down list to clone.
 when I start as a clone	When a sprite is cloned, this hat block runs in the newly made clone.
 delete this clone	This block deletes the clone it runs in and stops all its scripts.

- When the game begins, 10 fish appear, and each of them swims around on the stage.



```
when I receive start
repeat 10
    create clone of myself
    change num_fish by 1
```



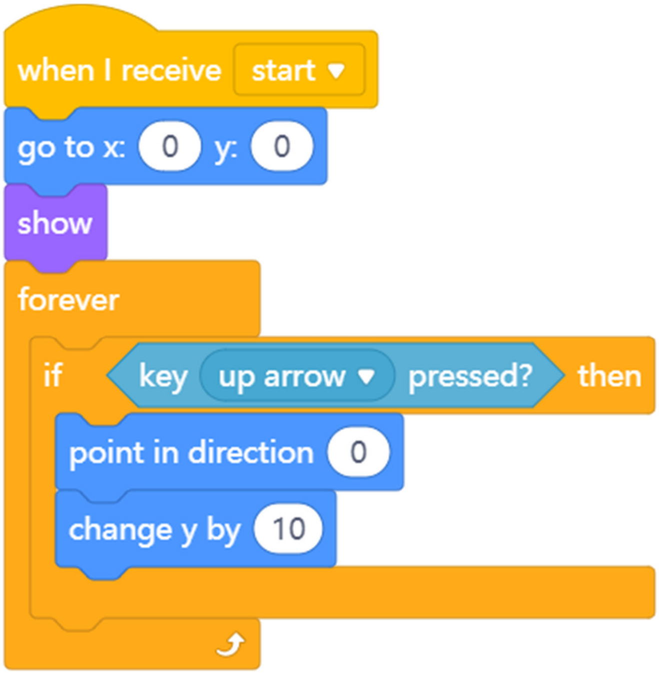
```
when I start as a clone
go to random position
show
point in direction pick random -180 to 180
forever
    move pick random 1 to 10 steps
    if on edge, bounce
```

- *NOTE:*

  *① Use the block **repeat ()** to decide the number of clones you need to create.*

  ② Use random numbers to set up the fish, including their location, swimming direction and speed.

- Press the arrow keys to move the fishing net:

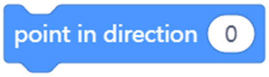
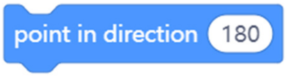
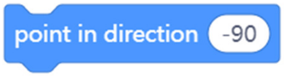
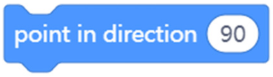


```
when I receive start ▾
go to x: 0 y: 0
show
forever
    if key up arrow ▾ pressed? then
        point in direction 0
        change y by 10
```

- ○ *NOTE:*

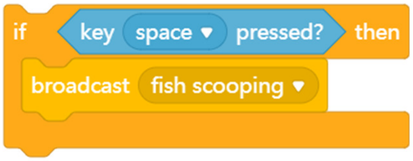
  ① The example script above programs the fishing net to move upward when the ↑ key is pressed.

  ② Similarly, complete the programs for the rest arrow keys. Refer to the table below:

Arrow keys	Point in direction	Coordinate	Increase by
↑	 point in direction 0	Y	10
↓	 point in direction 180	Y	-10
←	 point in direction -90	X	-10
→	 point in direction 90	X	10

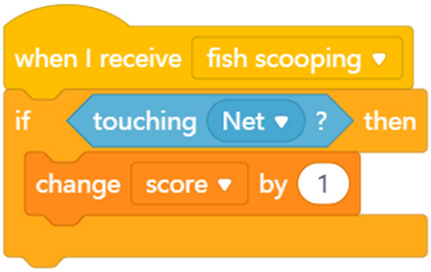
  *③ Modify the value in the block **change x/y by()** to set up the speed at which the fishing net is moving. A higher absolute value means a higher moving speed.*

- Press the space key to scoop the fish:



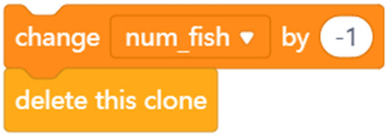
```
if <key [space ▾] pressed?> then
    broadcast [fish scooping ▾]
```

- Define the scoring system: the score increases by 1 each time fish is scooped:



```
when I receive [fish scooping ▾]
if <touching [Net ▾] ?> then
    change [score ▾] by (1)
```

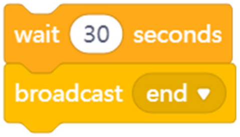
  - *NOTE: The program decides whether any fish is scooped successfully by detecting whether the fish touches the fishing net or not.*
- The fish scooped disappears: When the fish is scooped, it disappears from the screen.



```
change [num_fish ▾] by (-1)
delete this clone
```

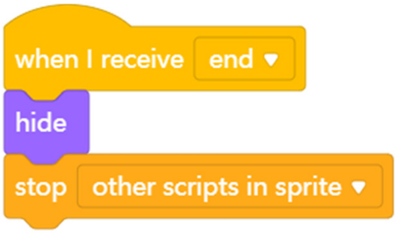
  - *NOTE: To make the fish disappear, just delete the sprite.*

- The 30-second countdown starts the moment the game begins. After 30 seconds, the game is over.



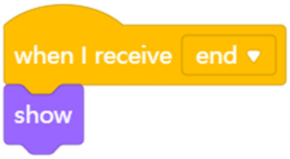
```
wait (30) seconds
broadcast [end ▾]
```

- The **Game Over** interface pops up when the game comes to an end. And we can see the final scores and a trophy on it. The fishing net is hidden.
  - Hide the fishing net:



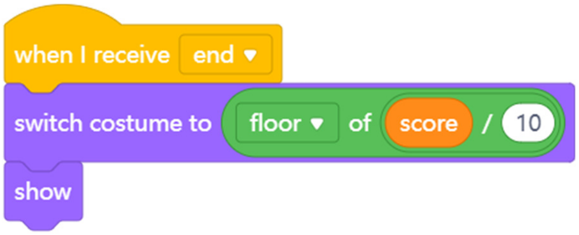
```
when I receive end ▾
hide
stop other scripts in sprite ▾
```

- Show a trophy:

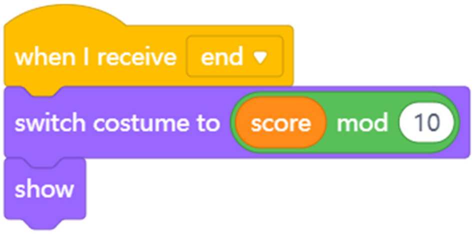


```
when I receive end ▾
show
```

- Show the final score:



```
when I receive end ▾
switch costume to floor ▾ of score / 10
show
```



```
when I receive end ▾
switch costume to score mod 10
show
```

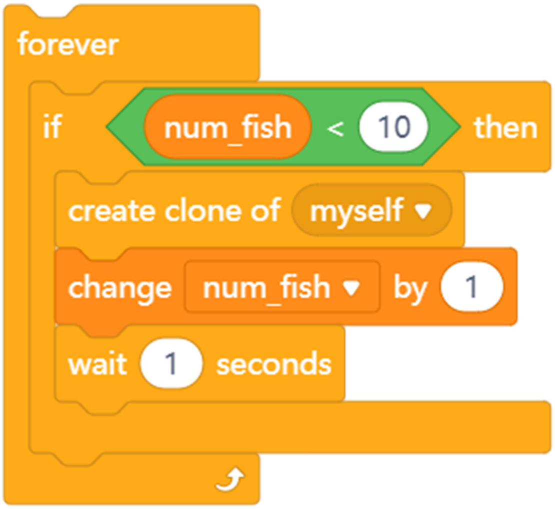
- *NOTE:*

  ① For the sprites "Ones" and "Tens", the number reported could be 0 to 9, and the numbers from 1 to 9, or 0 represents costumes 1-10 respectively.

  *② To calculate the number in tens place: Divide the score by 10 and if the result is a decimal number, then use the **(floor) of ()** block to round down the result. For example, when 36 is divided by 10, the result is 3.6. Round the result down to 3.*

  ③ To calculate the number in ones place: Divide the score by 10, and the remainder is the number in ones place. For example, when 36 is divided by 10, the remainder is 6 so the number in ones place is 6.

- Refill the pond with fish
  - If the player scoops all the fish on the stage, then nothing could be found on the stage for a while.
  - To ensure the stage has enough fish, write programs to enable this: more fish is added onto the stage whenever the total number of fish on the stage is less than 10.



```
forever
    if num_fish < 10 then
        create clone of myself ▾
        change num_fish ▾ by 1
        wait 1 seconds
```

Section 5 – Evaluate
[5 minutes]

Objective:

- Have students test their programs and games and compete with each to see who wins the highest score.

Procedures:

- Ask students to play their game with their desk mate or partner.
- Have 1-3 students share their experience and ask them what pros and cons the game has.

Lesson 12

Catch Fish Carnival Game II

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Identify the concept of **Motion-Controlled Games** and what features they have.
- Create a CyberPi controller with which they can change the movements of sprites.
- Track the operational data, and decide the result based on the data records.
- Program "Catch Fish Carnival Game II" in which players use CyberPi to interact with the game.

Key Focus

- Using CyberPi as a controller to move the sprites
- Tracking operational data in a game, and decision-making concerning the results based on the data records

Content Standards

Type	Indicator	Standard
CSTA	2-AP-11	Create clearly named variable that represent different data types and perform operations on their values.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
CSTA	2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
ISTE	5c	Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

Preparation

For the Teacher:

- A desktop or laptop with mBlock installed
- A CyberPi kit

For Students:

- Already know the role of data in showing game results, and the importance of scores and rankings in a game
- Already know the rules of Catch Fish Carnival Game, and completed the Catch Fish Carnival Game I project

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Introduce students to the concept of motion-controlled games.
5 minutes	**Section 2 – Explore** • Have 1 or 2 students try the game, and ask the rest to observe while they're playing. Ask students to figure out what's new about the game compared to Catch Fish Carnival Game I.
5 minutes	**Section 3 – Explain** • Have students use mind maps to analyze the mechanics of the game and organize their thoughts, preparing them for the later coding activities.
20 minutes	**Section 4 – Elaborate** • Give students freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Have students test each other's programs and games and compete to see who wins the highest score.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Introduce students to the concept of motion-controlled games.

Procedures:

- **Explain:** Motion-controlled games provide a new way for players to interact with games. Players get more immersed in games by physically interacting with games. Besides, this type of games requires players to move their body, so people have a chance to relax and get some exercise.

Section 2 – Explore
[5 minutes]

Objective:

- Have 1 or 2 students try the game and ask the rest to observe while they're playing. Ask students to figure out what's new about the game compared to Catch Fish Carnival Game I.

Procedures:

- **Introduce:** Last time, we interacted with the Catch Fish carnival game via a keyboard. Today, we're going to upgrade the game.
- Pick 1 or 2 students to try the game.
- **Start a class discussion:** Compare the upgraded version with the previous one. Are there any differences?
- **Summarize:** The upgraded version adopts a new control method which turns it into a motion-controlled game. Players interact with the game using the CyberPi. When the game is over, it gives you a star ranking and scores based on your success rate. A higher success rate can earn you a higher ranking and more points.

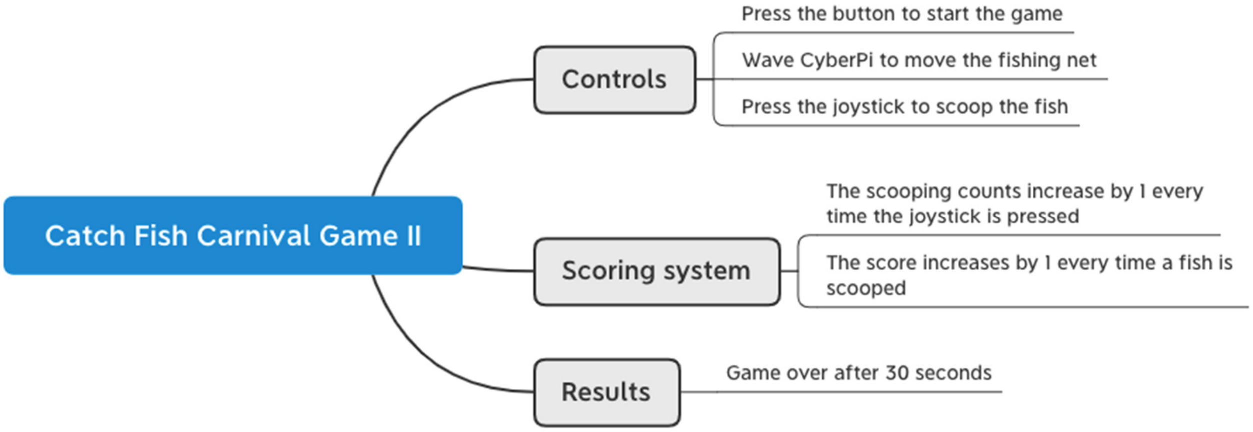
Section 3 – Explain
[5 minutes]

Objective:

- Have students use mind maps to analyze the mechanics of the game and organize their thoughts, preparing them for the later coding activities.

Procedures:

- Work with students to analyze what features Catch Fish Carnival Game II should have.
- Work with students to draw a mind map.


Catch Fish Carnival Game II

Controls
Press the button to start the game
Wave CyberPi to move the fishing net
Press the joystick to scoop the fish

Scoring system
The scooping counts increase by 1 every time the joystick is pressed
The score increases by 1 every time a fish is scooped

Results
Game over after 30 seconds

Section 4 – Elaborate
[20 minutes]

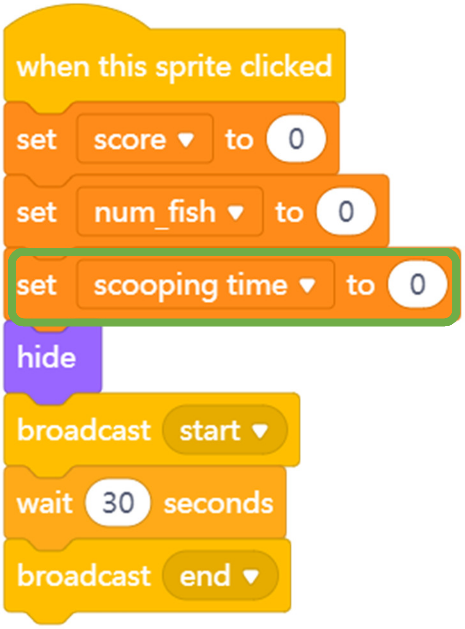
Objective:

- Give students freedom to explore further so that they can enhance their programming skills.

Procedures:

- Ask students to open the program file they previously created. If any of the students didn’t complete the program the previous lesson, send them yours.
- Add the device CyberPi. Connect CyberPi to mBlock, and switch to **Live** mode.

- Press the button to start the game
  - Make sure the data is cleared before the game starts.
  - The game decides the result based on the player's success rate. Create a variable to track the scooping times.

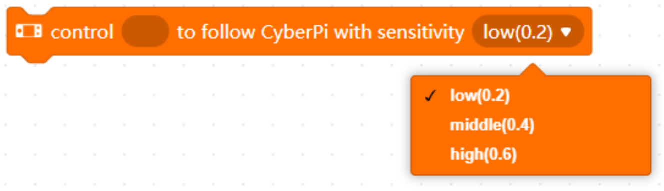


```
when this sprite clicked
set score to 0
set num_fish to 0
set scooping time to 0
hide
broadcast start
wait 30 seconds
broadcast end
```

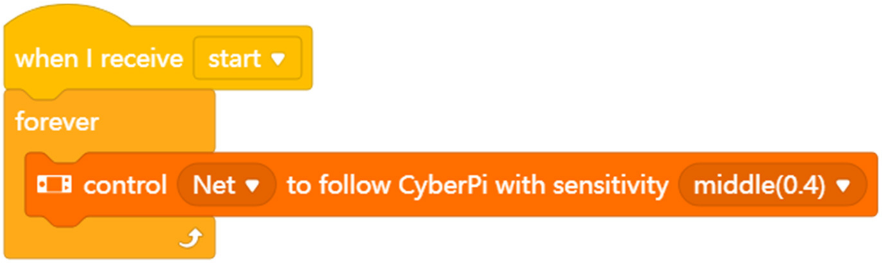
  - *NOTE:*

    Add the coding block to the previous script that's written to program the "Start" button to refresh data.

- Wave CyberPi to move the fishing net:
    - Introduce the motion-sensing block needed to turn CyberPi into a controller:

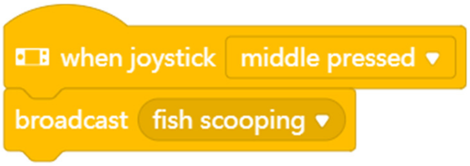
Coding block	Feature
 control ( ) to follow CyberPi with sensitivity low(0.2) ✓ low(0.2) middle(0.4) high(0.6)	Use this block to move the sprite with CyberPi. If you wave CyberPi at the same speed, then the sensitivity will determine the speed of the sprite. A higher sensitivity means a higher speed.

-
    - Use CyberPi to move the fishing net:



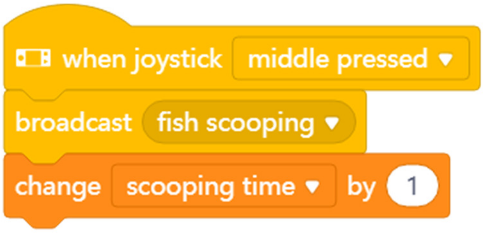
```
when I receive start
forever
    control Net to follow CyberPi with sensitivity middle(0.4)
```

- Press the joystick to scoop the fish.



```
when joystick middle pressed
broadcast fish scooping
```

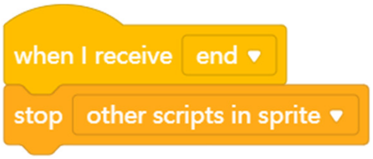
- When the joystick is pressed, the scooping count is increased by 1.



```
when joystick middle pressed
broadcast fish scooping
change scooping time by 1
```

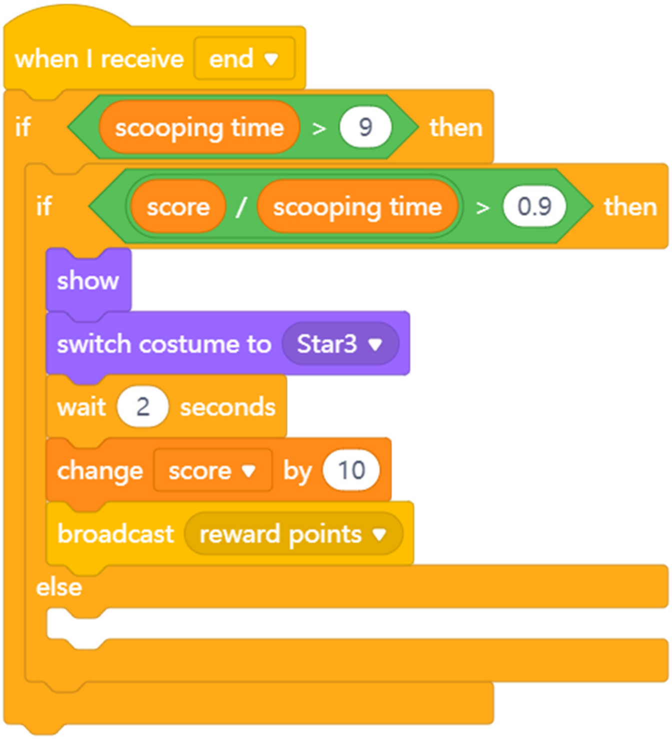
- When the fish is successfully scooped, the score goes up by 1: The scoring system is the same with "Catch Fish Carnival Game I" so, the script for this part doesn't need modification.

- Game is over when countdown stops



```
when I receive [end ▾]
stop [other scripts in sprite ▾]
```

- "Game Over" interface
  - Ranking: The game uses a star rating system and reward points to measure the player's performance based on the success rate. To raise the threshold, the player will be rated on a scale of stars only when the scooping count exceeds 9.
  - Success rate = number of fish scooped (score) / scooping count



```
when I receive [end ▾]
if <(scooping time) > (9)> then
    if <((score) / (scooping time)) > (0.9)> then
        show
        switch costume to [Star3 ▾]
        wait (2) seconds
        change [score ▾] by (10)
        broadcast [reward points ▾]
    else
```

  - *NOTE:*

    ① *The example above is the script for a 3-star rating.*

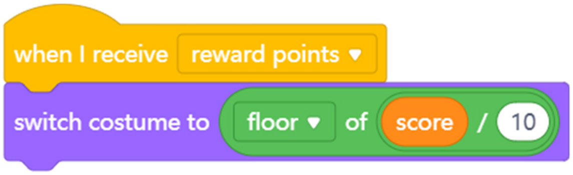
    ② *Refer to the following table to complete the programs for the whole star rating system.*

Success rate	Star rating	Reward points
90%~100%	3 stars	10
80%~90%	2 stars	6
70%~80%	1 star	2

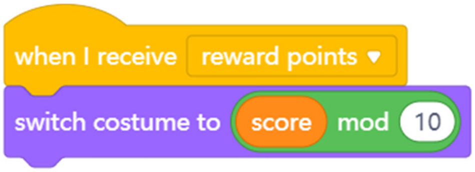
0~70%	None	None

- Display the final score:

  Add coding blocks to display the final score



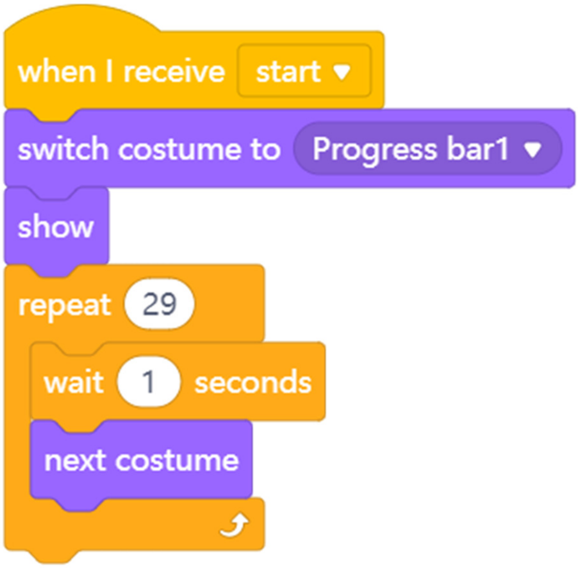
```
when I receive [reward points ▾]
switch costume to ([floor ▾] of ((score) / (10)))
```



```
when I receive [reward points ▾]
switch costume to ((score) mod (10))
```

- Have students test their program and consider any other effects they could add to the game.

- Program a progress bar to display the time left during the game

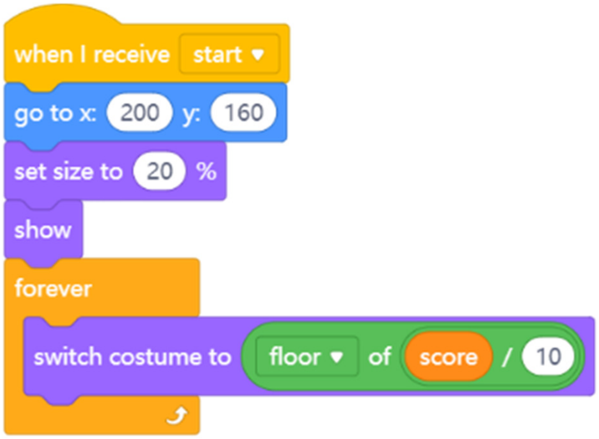


```
when I receive [start ▾]
switch costume to [Progress bar1 ▾]
show
repeat (29)
    wait (1) seconds
    next costume
```

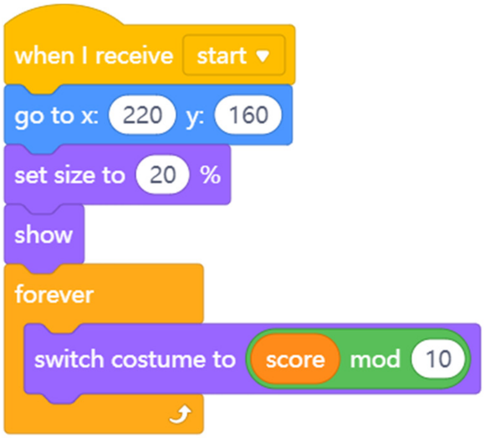
  - *NOTE:*

    The sprite "Progress Bar" has 30 costumes, so program the sprite to switch to another costume ever second.

- Display real-time scores



```
when I receive start ▾
go to x: 200 y: 160
set size to 20 %
show
forever
    switch costume to floor ▾ of score / 10
```



```
when I receive start ▾
go to x: 220 y: 160
set size to 20 %
show
forever
    switch costume to score mod 10
```

- *NOTE:*

  ① During the game or when the game is over, the score is visually different. So have students set up the position and size properly.

  ② Refer to the table below to set up the position and size:

Sprite	Position	Size
Tens	(x: -5, y: -5)	70
Ones	(x: 70, y: -5)	70

Section 5 – Evaluate
[5 minutes]

Objective:

- Have students test each other's programs and games.Let them compete to see who gets the highest score and wins.

Procedures:

- Have students invite their classmate or student partner to try their game and compete to see who gets the highest score and wins.
- Invite 1-3 students to share their experiences and come up with pros and cons.

Lesson 13

Dodge the Bird I

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

Objectives

By the end of class, students will be able to:

- Explain why and how player characters are created
- Draw a mind map to analyze the essential features of Dodge the Bird
- Use variables and create a prototype of Dodge the Bird

Key Focus

- Understanding the role and importance of data in player character selection
- Drawing a mind map to analyze the basic game effects of Dodge the Bird
- Completing the code for Dodge the Bird based on the mind map

Content Standards

Type	Indicator	Standard
CSTA	2-AP-11	Create clearly named variable that represent different data types and perform operations on their values.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
CSTA	2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
ISTE	5c	Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

Preparation

For the Teacher:

- A laptop or desktop with mBlock installed
- A CyberPi kit

For Students:

- Knowledge of how to use sensors
- Proficiency in block-based coding
- Knowledge of game design process and ability to design basic game mechanics

Agenda (40~45 minutes)

Duration	Content
5~10 minutes	**Section 1 – Engage** • Introduce player character design and its importance in a game.
5 minutes	**Section 2 – Explore** • Describe what the game is like to students and encourage them to brainstorm how to develop the game.
5 minutes	**Section 3 – Explain** • Work with students to draw a mind map and analyze the game mechanics, preparing students for the Elaborate session.
20 minutes	**Section 4 – Elaborate** • Give students the freedom to explore further so that they can enhance their programming skills.
5 minutes	**Section 5 – Evaluate** • Allow students time to play the game that they've developed and have them improve the game by considering user preference.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Introduce player character design and its importance in a game.

Procedures:

- **Ask:** Some games offer different playable characters for players to choose from. Have you ever played any such games?
  - Encourage students to describe the games in detail. Guide them with questions like, "What's the name of the game?" and "What kind of player characters are available to choose from?"
- **Summarize:** Quite a few games allow players to choose the character they want to control.
- **Ask:** As a player, what do you think about this feature? What if you didn't have the option to choose a character?
- **Summarize:** Providing different characters for players to choose from enriches a game and the player experience but also sustains players' motivation and interest in playing the game.

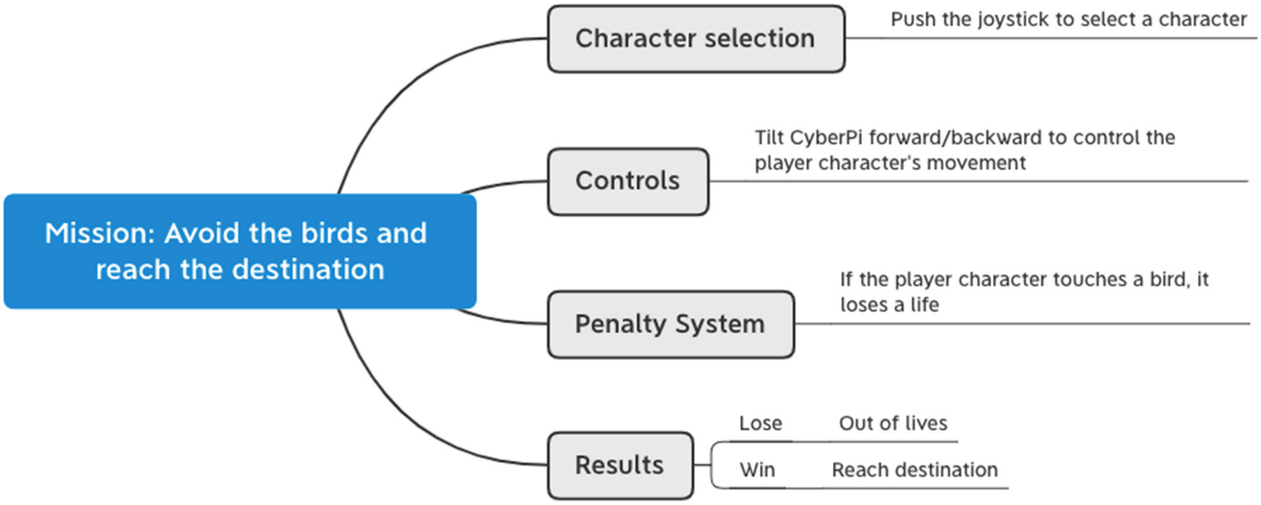
Section 2 – Explore
[5 minutes]

Objective:

- Describe what the game is like to students and encourage them to brainstorm how to develop the game.

Procedures:

- Invite volunteers to try Dodge the Bird. Have them pay attention to the player character selection page.
- **Ask:** How did the player choose a character in Dodge the Bird? What's the logic behind the player character selection system? (Hint: if…then…) And describe how to play the game.
- **Summarize:** In the game, the player uses CyberPi's joystick to select a character. Push the joystick to the right to select Witch and push the joystick to the left to select Jinnee. Let's take a look at the gameplay of Dodge the Bird:


Mission: Avoid the birds and reach the destination
Character selection
Push the joystick to select a character
Controls
Tilt CyberPi forward/backward to control the player character's movement
Penalty System
If the player character touches a bird, it loses a life
Results
Lose
Out of lives
Win
Reach destination

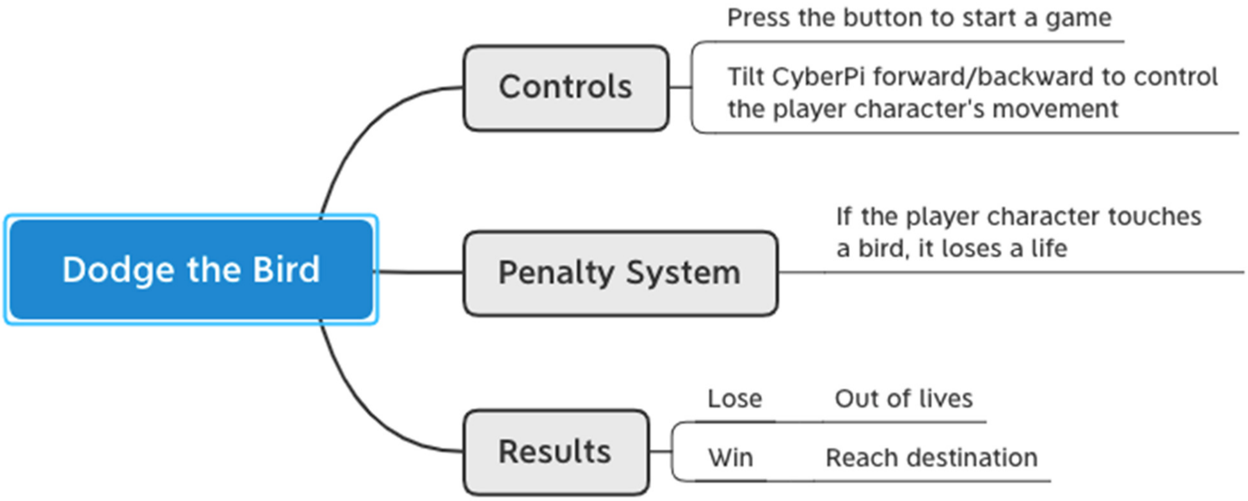
Section 3 – Explain
[5 minutes]

Objective:

- Work with students to draw a mind map and analyze the game mechanics, preparing students for the Elaborate session.

Procedures:

- Work with students to analyze some features of *Dodge the Bird*.
- Instruct students to draw a mind map.


Dodge the Bird
Controls
Press the button to start a game
Tilt CyberPi forward/backward to control the player character's movement
Penalty System
If the player character touches a bird, it loses a life
Results
Lose
Out of lives
Win
Reach destination

Section 4 – Elaborate
[20 minutes]

Objective:

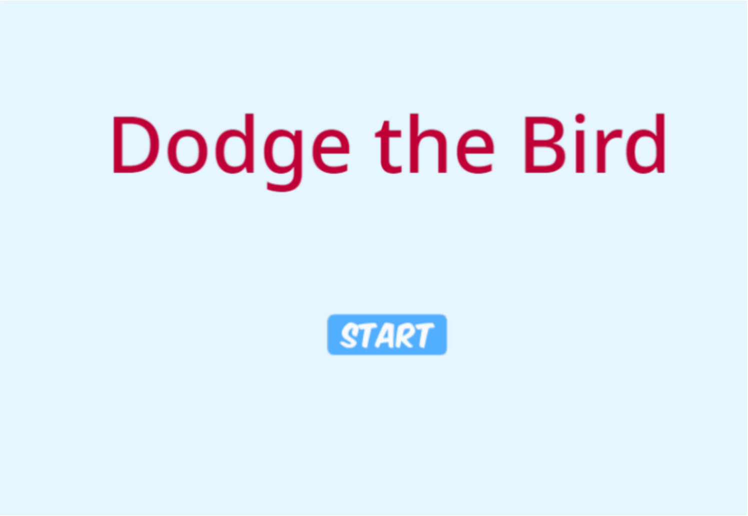
- Give students the freedom to explore further so that they can enhance their programming skills.

Procedures:

- Connect CyberPi to mBlock and select the **Live** mode.
- Open the demo program file and go through it. The file includes preset programs for:

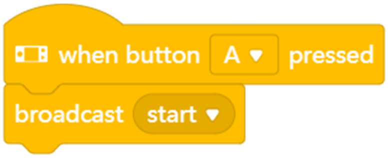
Sprite	Feature
	1. When the program starts running, the cloud floats.
Dodge the Bird	1. When the green flag is clicked, the title and instructions appear. 2. When the game starts, the title and instructions disappear.
	1. When the green flag is clicked, the sign "Start" appears. 2. When the game starts, the sign disappears.
	1. When the game starts, birds fly from right to left on the stage. 2. When touching the player character, the bird disappears.

- Click the green flag to enter the startup interface:



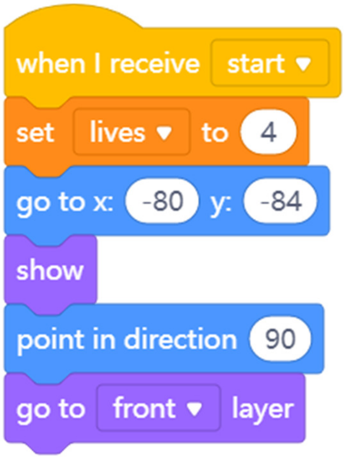
Dodge the Bird

  - The startup interface has been preset with the following script:



```
when button A ▾ pressed
broadcast start ▾
```

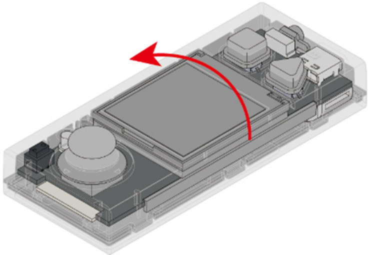
- Define the initial status of a player character:
  - Set the initial status of player characters, including location and direction.
  - Create a variable "lives" and set it to 4. This setting gives a player character four lives. The variable will be later used in the script to determine whether a player loses the game.



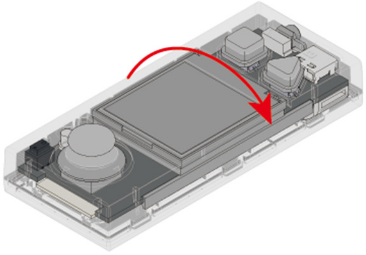
```
when I receive start ▾
set lives ▾ to 4
go to x: -80 y: -84
show
point in direction 90
go to front ▾ layer
```

- Control characters:
  - Characters move up/down: Tilt CyberPi to make a player character move up or down.

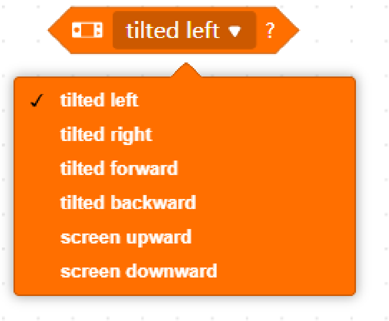
Tilted forward:



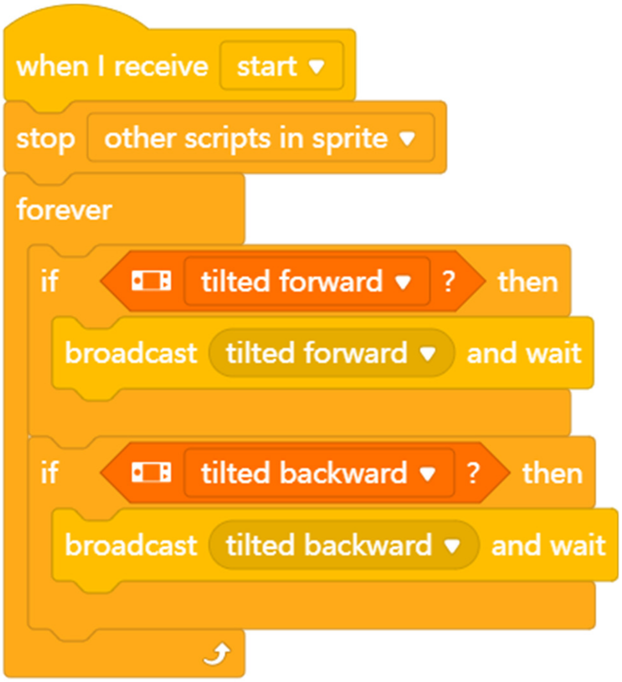
Tilted backward:



  - Use the Motion Sensing block in the table to detect which direction CyberPi is tilted to:

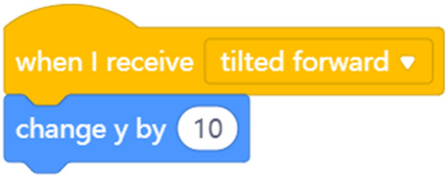
Coding block	Feature
 tilted left ▾ ? ✓ tilted left tilted right tilted forward tilted backward screen upward screen downward	This Boolean block offers 6 options that can be programmed to control different effects.

- CyberPi tilted forward/backward:

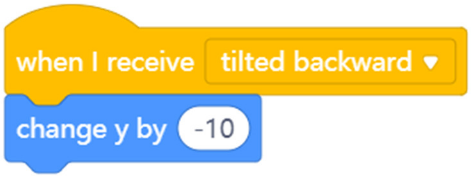


```
when I receive start ▾
stop other scripts in sprite ▾
forever
    if tilted forward ▾ ? then
        broadcast tilted forward ▾ and wait
    if tilted backward ▾ ? then
        broadcast tilted backward ▾ and wait
```

- *NOTE:*
    - Wrap the "tilted forward?" and "tilted backward?" blocks into the same **forever** block.
- Sprite moves:



```
when I receive tilted forward ▾
change y by 10
```

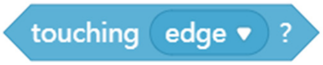


```
when I receive tilted backward ▾
change y by -10
```

- Prevent sprites from moving out of the stage

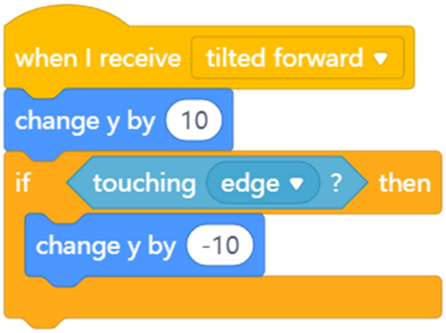
  If a sprite keeps going up or down, it will disappear from the stage.

To avoid a sprite disappearing, add this sensing block to the script:

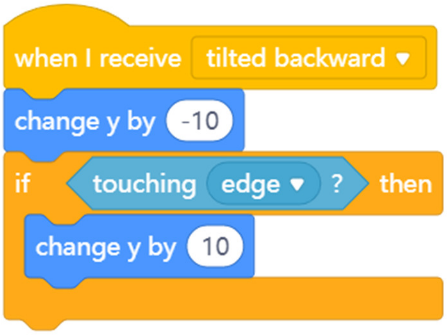


```
touching edge ▾ ?
```

Write these scripts:



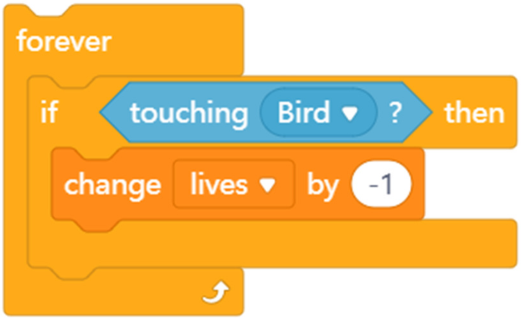
```
when I receive tilted forward ▾
change y by 10
if touching edge ▾ ? then
    change y by -10
```



```
when I receive tilted backward ▾
change y by -10
if touching edge ▾ ? then
    change y by 10
```

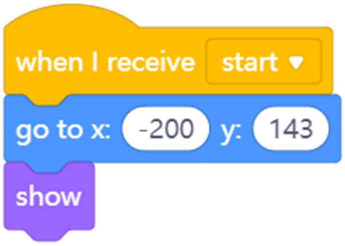
- Both scripts involve two **change y by ()** blocks. Take the script on the left as an example. When receiving "Up", the sprite's y-coordinate changes by 10. To prevent the sprite from disappearing, type -10 in the second **change y by ()** block. The second **change y by ( )** block will offset the effect of the first one when the sprite touches the edge of the stage.

- Set up the penalty system
    - If the sprite touches the bird, it loses a life:



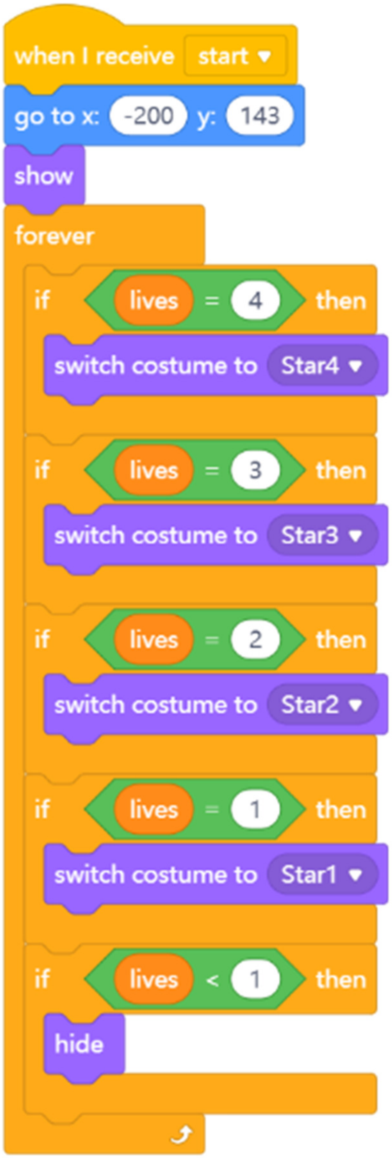
```
forever
    if touching Bird ▾ ? then
        change lives ▾ by -1
```

    - Add an animated effect to the life sprite:
        - To show how many lives the player has left, add a 4-star sprite at the top left corner of the stage. A star represents a life.
        - The 4-star sprite only appears when the game starts. So, move the sprite to a specified location but hide it at the very beginning.



```
when I receive start ▾
go to x: -200 y: 143
show
```

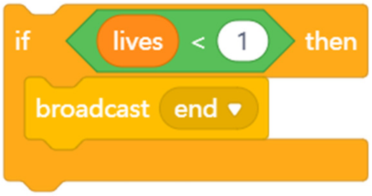
- The 4-star sprite has four costumes and switches costumes to tell the player how many lives are left. When "Lives" equals 0, the sprite disappears from the stage:



```
when I receive start ▾
go to x: -200 y: 143
show
forever
  if lives = 4 then
    switch costume to Star4 ▾
  if lives = 3 then
    switch costume to Star3 ▾
  if lives = 2 then
    switch costume to Star2 ▾
  if lives = 1 then
    switch costume to Star1 ▾
  if lives < 1 then
    hide
```

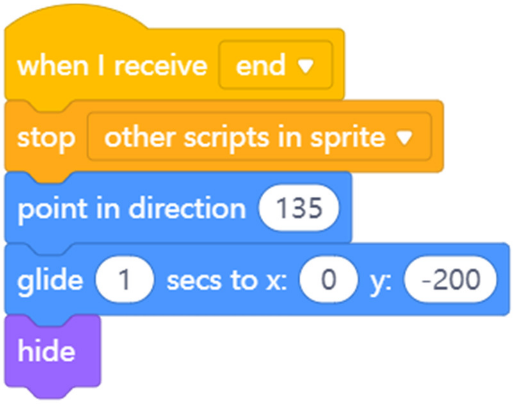
- *NOTE: Here's just an example program. Complete the script to make the sprite change costumes accordingly when "Lives" equals 3, 2, and 1.*

- Result—game over:



```
if <(lives) < (1)> then
    broadcast (end ▾)
```

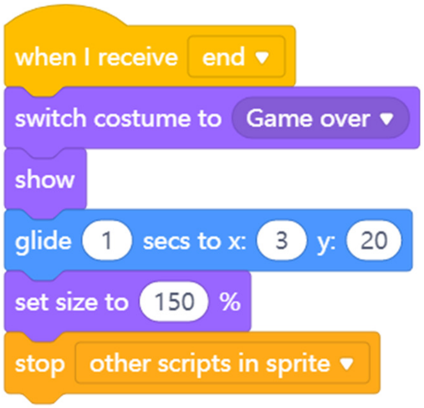
- Game-over effect:
    - Character falling: When the game is over, the player character stops flying and falls down. Use **point in direction ( )** and **glide ( ) secs to x: ( ) y: ( )** to mimic the falling effect.



```
when I receive (end ▾)
stop (other scripts in sprite ▾)
point in direction (135)
glide (1) secs to x: (0) y: (-200)
hide
```

-
    - Display "Game Over":
        - A "Game Over" sign rises from the bottom to the center of the stage.
        - Highlight the sprite "Signs" and click **Costumes**. One of the costumes is "Game Over".

-
    -
        - Make the "Game Over" sign appear at the center of the stage:



```
when I receive end ▾
switch costume to Game over ▾
show
glide 1 secs to x: 3 y: 20
set size to 150 %
stop other scripts in sprite ▾
```

Section 5 – Evaluate
[5 minutes]

Objective:

- Allow students time to play the game that they've developed and have them improve the game by considering user preferences.

Procedures:

- Invite two volunteers to try the game and compete to see who can survive the longest.
- Have students discuss: As a player, do you want other features or effects? What are they?

Lesson 14

Dodge the Bird II

Subject Area: Computing **Level: Introductory** **Time Frame: 45 minutes**

Ages: 11~13 years old **Year Groups: 6–8**

★ Objectives

By the end of class, students will be able to:

- Explain the roles of data in player character design
- Develop a game that allows players to choose a player character
- Draw a mind map to analyze the features of Dodge the Bird
- Use variables and complete the advanced version of Dodge the Bird

Key Focus

- Understanding the role and importance of data in game design
- Drawing a mind map to analyze the basic effects of Dodge the Bird
- Completing the code based on the mind map

Content Standards

Type	Indicator	Standard
CSTA	2-AP-11	Create clearly named variable that represent different data types and perform operations on their values.
CSTA	2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
CSTA	2-AP-13	Design projects that combine hardware and software components to collect and exchange data.
ISTE	5c	Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

Preparation

For the Teacher:

- A laptop or desktop with mBlock installed
- A CyberPi kit

For Students:

- Knowledge of how to use sensors
- Proficiency in block-based coding
- Knowledge of game design process and ability to develop simple game mechanics
- A prototype of Dodge the Bird

Agenda (40~45 minutes)

Duration	Content
5 minutes	**Section 1 – Engage** • Recap the previous lesson and clarify the objectives of this lesson.
10 minutes	**Section 2 – Explain** • Instruct students to analyze a project with a mind map, preparing them for the programming session.
20 minutes	**Section 3 – Elaborate** • Give students the freedom to explore further so that they can enhance their programming skills.
5~10 minutes	**Section 4 – Evaluate** • Allow students time to try the game that they developed and have them think about the applications of data in other games.

Activities

Section 1 – Engage
[5~10 minutes]

Objective:

- Recap the previous lesson and clarify the objectives of this lesson.

Procedures:

- Show students the complete version of Dodge the Bird.
- **Summarize:** Last time we completed the following features:
    - control of a player character's movement
    - a penalty system: if a player character touches a hazard, it loses a life
    - a decision structure: if a player character is out of lives, the game ends

  We'll continue developing the game, adding a feature that allows players to choose a player character.

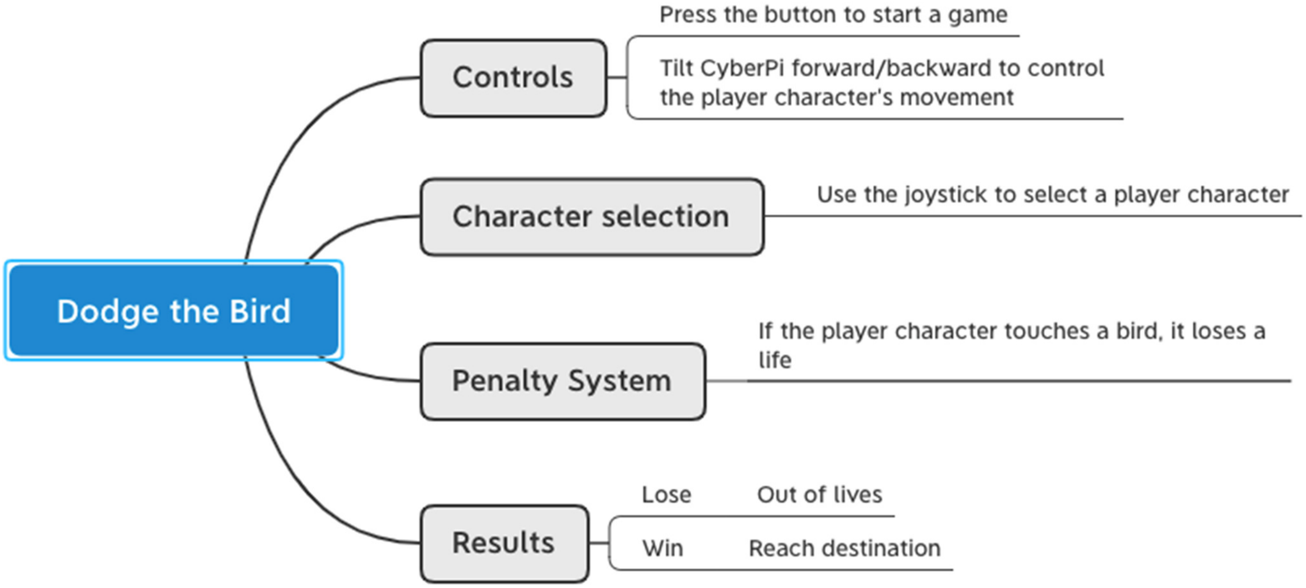
Section 2 – Explain
[10 minutes]

Objective:

- Instruct students to analyze a project with a mind map, preparing them for the programming session.

Procedures:

- Work with students to analyze the features of Dodge the Bird.
- Instruct students to draw a mind map.


Dodge the Bird
Controls
Press the button to start a game
Tilt CyberPi forward/backward to control the player character's movement
Character selection
Use the joystick to select a player character
Penalty System
If the player character touches a bird, it loses a life
Results
Lose
Out of lives
Win
Reach destination

Section 3 – Elaborate
[20 minutes]

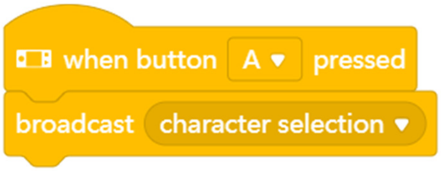
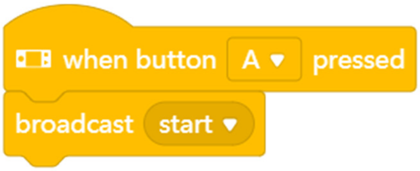
Objective:

- Give students the freedom to explore further so that they can enhance their programming skills.

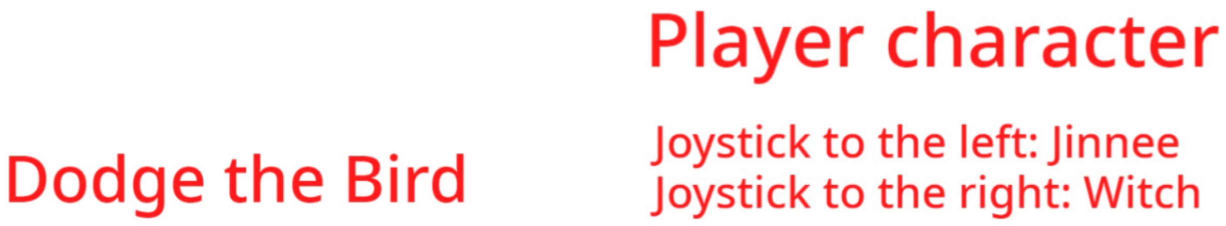
Procedures:

- Open the preset program that has the basic game features.
- Connect CyberPi to mBlock.

- Design a player character selection page:
  - In the programs we wrote during the previous lesson, the game begins as soon as we press the start button.
  - Let's make a small change to the program. When we press the button, we enter a player character selection page.

Example program (CyberPi)	Old version (CyberPi)
 when button A ▾ pressed broadcast character selection ▾	 when button A ▾ pressed broadcast start ▾

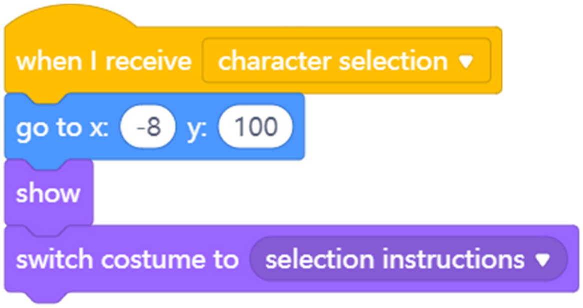
  - The player character selection page includes two parts: instructions and player characters.
    - Instructions: Click the sprite "Title and instructions" and find the costume "selection instructions".



Dodge the Bird

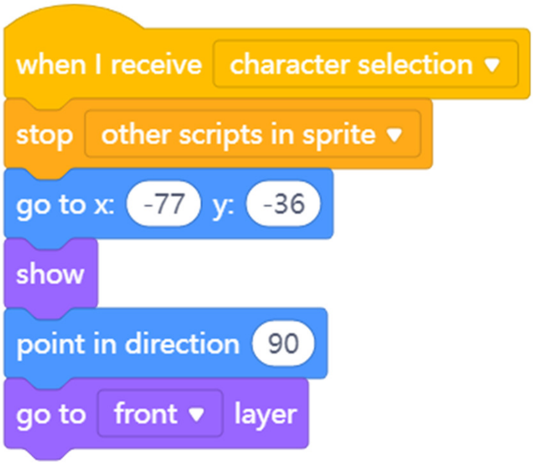
Player character

Joystick to the left: Jinnee
Joystick to the right: Witch

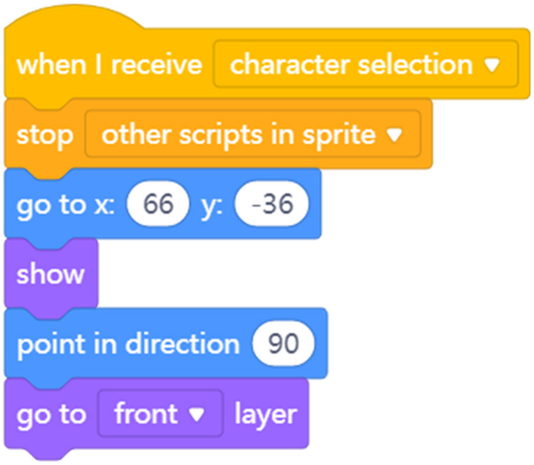


```
when I receive [character selection ▾]
go to x: (-8) y: (100)
show
switch costume to [selection instructions ▾]
```

- Character options: Display the player characters, Jinnee and Witch, on the stage.



```
when I receive [character selection ▾]
stop [other scripts in sprite ▾]
go to x: (-77) y: (-36)
show
point in direction (90)
go to [front ▾] layer
```

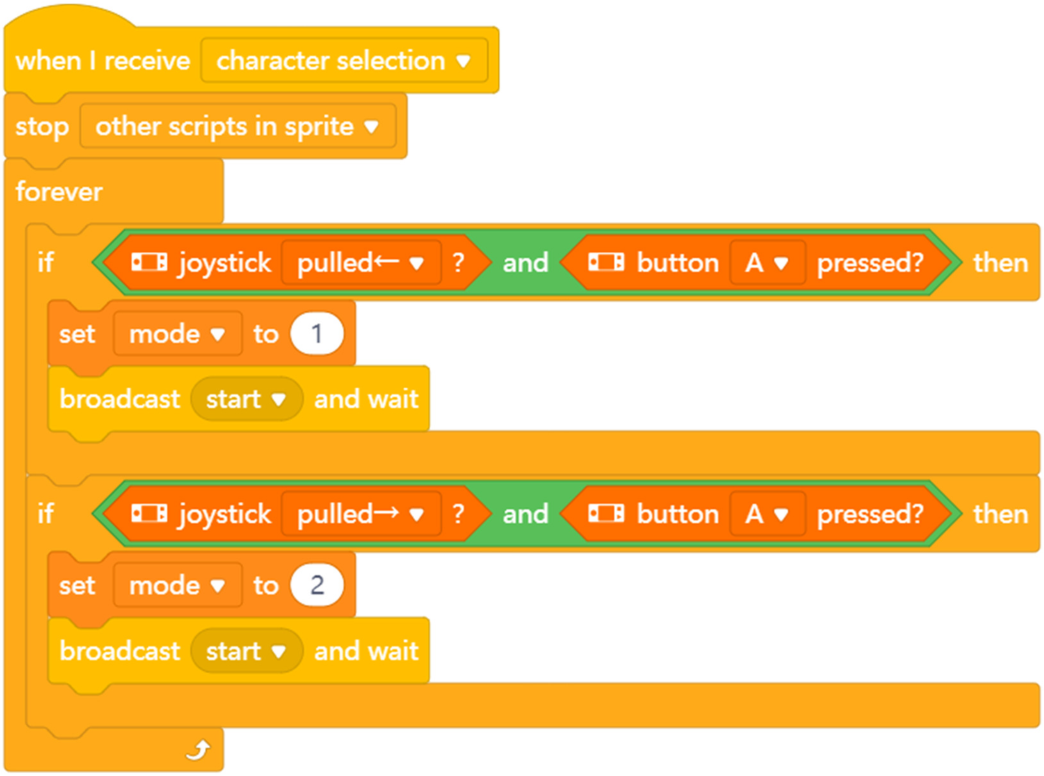


```
when I receive [character selection ▾]
stop [other scripts in sprite ▾]
go to x: (66) y: (-36)
show
point in direction (90)
go to [front ▾] layer
```

- *NOTE: To make sure every game has a fresh start, use the **stop other scripts in this sprite** block.*

- Select a player character:
  - Joystick to the left: Jinnee

    Joystick to the right: Witch.
  - To make it easier to extend the character selection feature, create a variable "Mode" to record players' options. "1" is Jinnee is and "2" Witch.



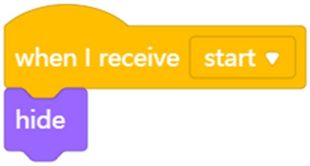
```
when I receive [character selection ▾]
stop [other scripts in sprite ▾]
forever
    if < joystick [pulled← ▾] ? > and < button [A ▾] pressed? > then
        set [mode ▾] to (1)
        broadcast [start ▾] and wait
    if < joystick [pulled→ ▾] ? > and < button [A ▾] pressed? > then
        set [mode ▾] to (2)
        broadcast [start ▾] and wait
```

- *NOTE:*

  *① To make sure every game has a fresh start, use **the stop other scripts in this sprite** block.*

  *② To make sure a player can't select characters after the game begins, use **broadcast ( ) and wait**.*

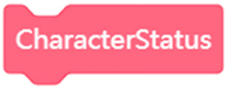
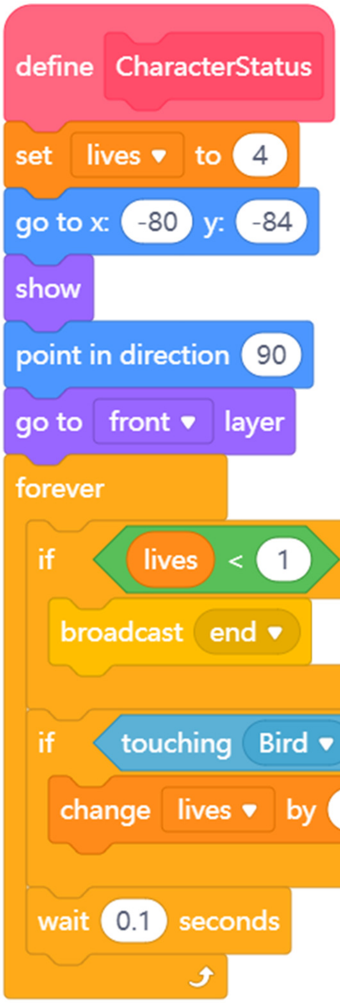
- After the player confirms the player character, the game starts.
- The "Title and instructions" sprite disappears when the game starts:



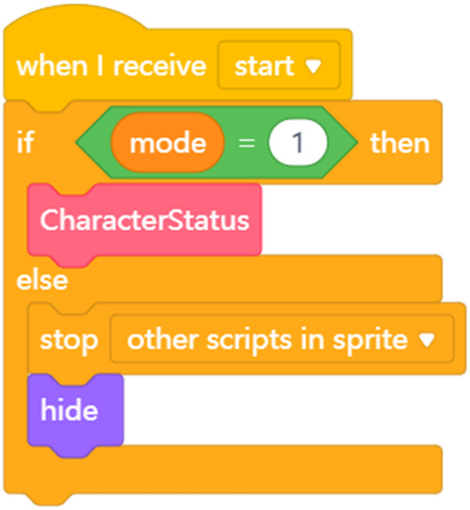
```
when I receive [start ▾]
hide
```

- The character chosen by the player appears on the stage. That is, when selecting Jinnee, the player continues the game as Jinnee. Build on the program that was created in the previous lesson.
  - To make the program easier to understand, create a function "CharacterStatus" to record a character's status after the game starts:

Function	Definition

 CharacterStatus	 define CharacterStatus set lives ▾ to 4 go to x: -80 y: -84 show point in direction 90 go to front ▾ layer forever if lives < 1 then broadcast end ▾ if touching Bird ▾ ? then change lives ▾ by -1 wait 0.1 seconds

- Include the function "CharacterStatus" and the variable "Mode" in the script:



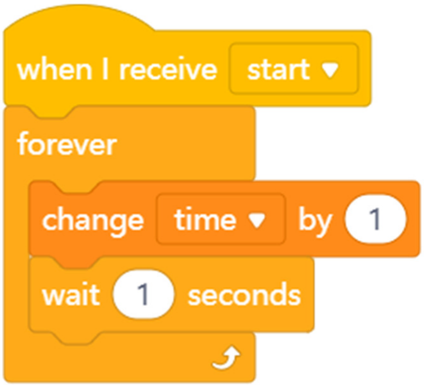
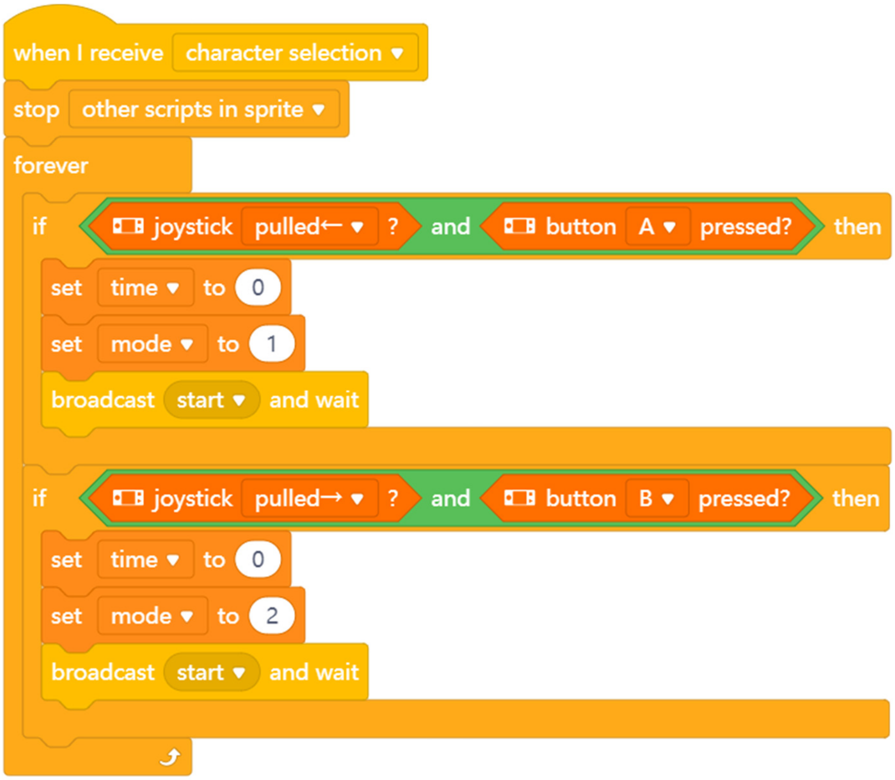
```
when I receive start ▾
if mode = 1 then
    CharacterStatus
else
    stop other scripts in sprite ▾
    hide
```

- *NOTE:*

① Based on the scripts for Jinnee, complete the ones for Witch, including an animated effect, player character status, up/down movement, and a game-over effect.

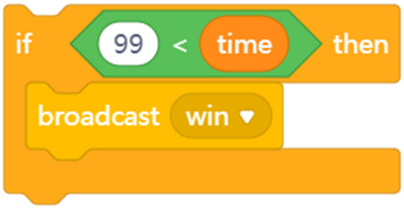
② Functions can't be shared across sprites. So, duplicate the scripts into the sprite Witch and then create and define the function in the scripts area of Witch.

- Result—Win: Last time, we completed the structure that decides whether a player loses the game. Now let's add another decision structure to tell whether a player wins. Set the game length to 100 seconds. If the character stays alive for 100 seconds, it means the character reaches the destination and the player wins.
  - ○ Make decisions based on time: Create a variable "time" to record the length of each game.

Example program (Sprite Sign)	Example program (CyberPi)
 when I receive start ▾ forever change time ▾ by 1 wait 1 seconds	 when I receive character selection ▾ stop other scripts in sprite ▾ forever if joystick pulled← ▾ ? and button A ▾ pressed? then set time ▾ to 0 set mode ▾ to 1 broadcast start ▾ and wait if joystick pulled→ ▾ ? and button B ▾ pressed? then set time ▾ to 0 set mode ▾ to 2 broadcast start ▾ and wait

- *NOTE: You can add the script that records the game length to any sprite, but it's better to put it in a sprite that has fewer scripts. In the example program, the length-recording script is under the sprite "sign".*

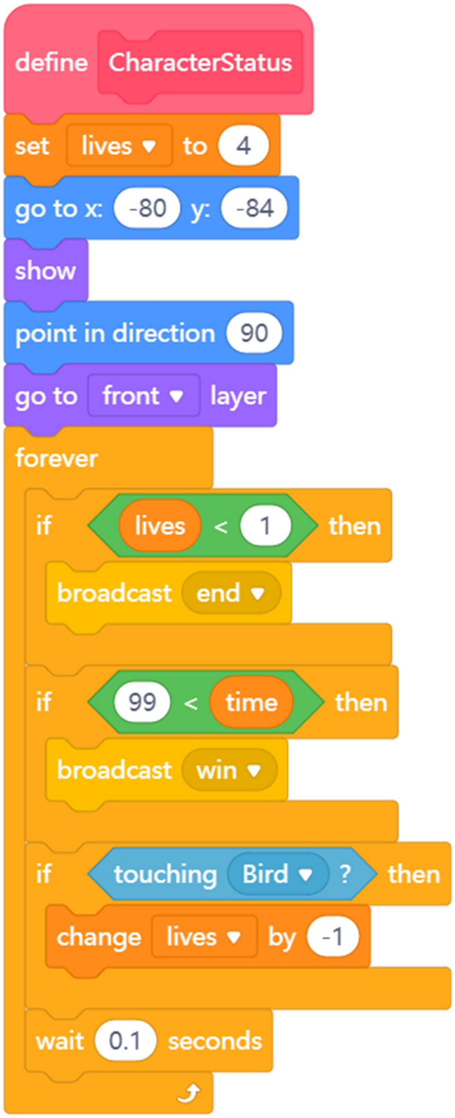
- ○ Decide whether a player wins:



```
if <(99) < (time)> then
    broadcast (win ▾)
```

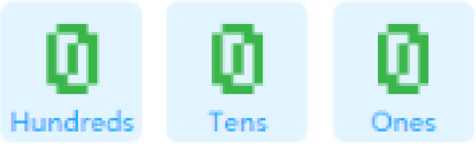
- *NOTE:*

  ① Join the blocks for deciding the result as shown:



```
define CharacterStatus
set (lives ▾) to (4)
go to x: (-80) y: (-84)
show
point in direction (90)
go to (front ▾) layer
forever
    if <(lives) < (1)> then
        broadcast (end ▾)
    if <(99) < (time)> then
        broadcast (win ▾)
    if <touching (Bird ▾) ?> then
        change (lives ▾) by (-1)
    wait (0.1) seconds
```

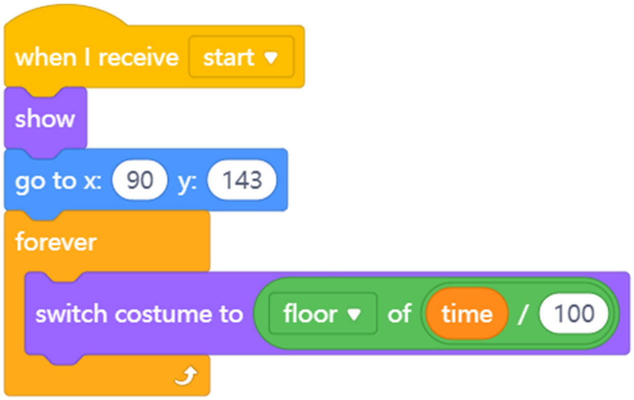
② Animated timer: Displaying the game time on the stage helps players know the game progress in real time. Do you still remember how we displayed scores in Fish Carnival Game? Use the same method to display the game time.



For the sprite **"Hundreds"**, divide the time by 100 and round the result down. For example,

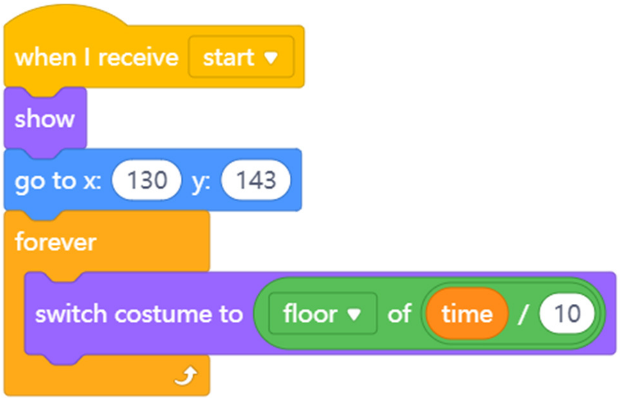
120/100=1.2, round 1.2 down to 1

260/100=2.6, round 2.6 down to 2



```
when I receive [start ▾]
show
go to x: (90) y: (143)
forever
    switch costume to ([floor ▾] of ((time) / (100)))
```

Do the similar with the sprite **"Tens"**, dividing the time by 10 and round down the result:

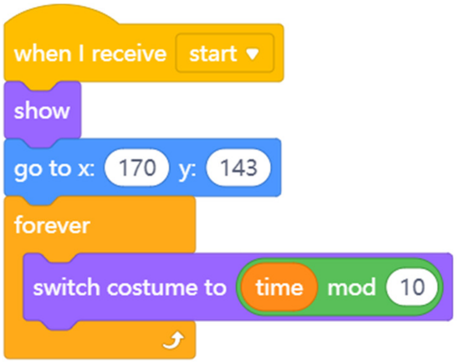


```
when I receive [start ▾]
show
go to x: (130) y: (143)
forever
    switch costume to ([floor ▾] of ((time) / (10)))
```

- *NOTE:*

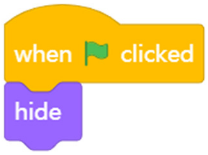
① Take "130/10=13" as an example. The result is 13 but the sprite has only 10 looks. So, the sprite turns to its third look, "3".

*② **As for** the sprite "Ones", use **( ) mod ( )**. For example, 4/10=0.4, and take the remainder "4".*

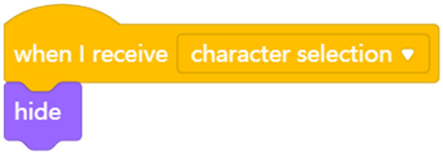


```
when I receive start ▾
show
go to x: 170 y: 143
forever
    switch costume to (time mod 10)
```

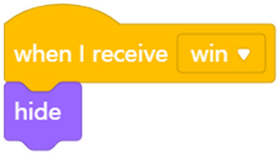
Based on the game effects, these digits (hundreds, tens, and ones) don't appear on the startup, character selection, and victory interfaces, and they stop updating when the game ends. To achieve these effects, write the following scripts:



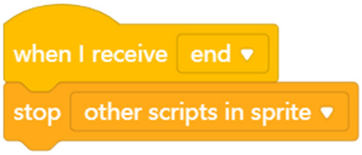
```
when ⚑ clicked
hide
```



```
when I receive character selection ▾
hide
```

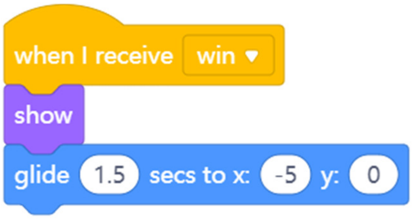


```
when I receive win ▾
hide
```



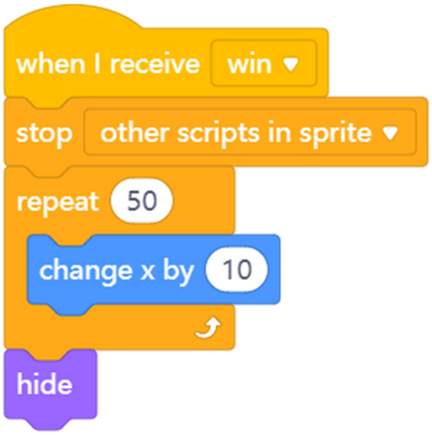
```
when I receive end ▾
stop other scripts in sprite ▾
```

- Now, add the victory effects:
  - ○ Destination appears:



```
when I receive win ▾
show
glide 1.5 secs to x: -5 y: 0
```

- Character animation:



```
when I receive [win ▾]
stop [other scripts in sprite ▾]
repeat (50)
    change x by (10)
end
hide
```

- *NOTE: The example programs are for the Sprite Jinnee. Duplicate the scripts into Witch.*

Section 4 – Evaluate
[5~10 minutes]

Objective:

- Allow students time to try the game that they developed and have them think about the applications of data in other games.

Procedures:

- Invite two students to try the game and let them compete to see who can finish the game.
- **Explain:** *Dodge the Bird* needs data to complete its features. Whether CyberPi’s button is pressed and whether CyberPi is tilted forward or backward are all reported to the program as data. We used variables, like "Lives" and "Time" to keep a record of all the data we want. And the major feature, player character selection, relies on the data stored in the variables to call upon the corresponding scripts.
- **Ask:** Think about the games you’ve played. Which features of them may involve data?