Lesson Plan 1: Controlling a Character

Subject: STEM / Language Arts / Computer Science / Computational Thinking
Duration: 60 minutes

Learning Objectives:

- Students will understand the concept of controlling a character using keyboard inputs.
- Students will learn how to write simple commands to move a character in different directions.
- Students will practice writing sentences that describe character movements based on user inputs.

Introduction (10 minutes):

1. **Hook:** Start with a brief discussion about video games and animations. Ask students if they have ever wondered how characters in games move when they press certain keys.
2. **Objective:** Explain that today they will learn how to control a character using simple commands and write sentences to describe these actions.

Direct Instruction (15 minutes):

1. **Concept Explanation:**

   - Introduce the concept of keyboard inputs and how they can be used to control a character.
   - Explain the basic commands for moving a character (e.g., up, down, left, right).
   - Show examples of sentences that describe these actions, such as "When the up arrow is pressed, the rabbit moves up."

2. **Demonstration:**

   - Use a projector or interactive whiteboard to demonstrate how to create a simple program that moves a character using keyboard inputs.
   - Write example commands and sentences on the board:

```
There is a rabbit.

When the up arrow is pressed, the rabbit moves up.

When the down arrow is pressed, the rabbit moves down.

When the left arrow is pressed, the rabbit moves left.

When the right arrow is pressed, the rabbit moves right.
```

Guided Practice (20 minutes):

1. **Hands-On Activity:**

   - Have students go on to Game Changineer
   - Guide them through the process of adding a character, such as “There is a rabbit.”
   - Encourage students to write sentences describing the actions they are programming.

2. **Monitoring and Support:**

   - Walk around the classroom to provide assistance and answer questions.
   - Check that students are correctly writing the commands and corresponding sentences.

Independent Practice (10 minutes):

1. **Challenge:**
   - Ask students to use a different character instead of rabbit (they can pick any, such as fox).
   - Ask the students move the character using W A S D instead of arrow keys.
   - Hint: When W is pressed, the fox moves up.

2. **Challenge 2:**
   - What would the following sentence do? What error message(s) are generated?

```
The rabbit moves up with the up arrow.
```

   - How would you fix the sentence? Hint: think about starting the sentence with “When …”

Closure (5 minutes):

1. **Review:**

   - Recap the key points of the lesson: controlling a character with keyboard inputs and writing descriptive sentences.
   - Ask a few students to share their sentences and demonstrate their projects.

2. **Homework (Optional):**

   - Assign students to include 2 characters (for example, rabbit and fox), and one character is controlled with the arrow keys while the other is controlled with W A S D.

Assessment:

- Observe students during the hands-on activity to ensure they understand the concepts.
- Review the sentences students write to check for accuracy and clarity.

Helpful Points to Watch For:

- Ensure students understand the relationship between the command and the action (e.g., pressing the up arrow makes the character move up).
- Encourage students to use clear and descriptive language in their sentences.
- Provide support for students who may struggle with the programming environment or writing sentences.

Lesson Plan: 2 - Cause and Effect in Programming

Subject: STEM / Language Arts / Computer Science / Computational Thinking
Duration: 60 minutes

Learning Objectives:

- Students will understand the concept of cause and effect in programming.
- Students will learn how to create events that trigger actions in a program.
- Students will practice writing sentences that describe cause-and-effect relationships in their code.

Introduction (10 minutes):

1. **Hook:** Start with a brief discussion about cause and effect in everyday life. Ask students for examples, such as "If I touch a hot stove, my hand will get burned."
2. **Objective:** Explain that today they will learn how to create cause-and-effect relationships in programming, using simple commands.

Direct Instruction (15 minutes):

1. **Concept Explanation:**

   - Introduce the concept of events in programming and how they can trigger actions.
   - Explain the basic idea of cause and effect: one action (cause) leads to another action (effect).
   - Cause is also called ‘antecedent’, and the effect also known as ‘consequent’
   - Show examples of sentences that describe these relationships, such as "When the rabbit touches a carrot, the carrot disappears."
   - When there are pronouns, discuss how to interpret the pronoun. Sometimes it might be ambiguous (not knowing which character the pronoun refers to)
   - How to write clearly – it may be necessary to replace the pronoun with the actual character. Example: When the rabbit touches a carrot, it disappears. (Replace ‘it’ with ‘the carrot’)

2. **Demonstration:**

   - Use a projector or interactive whiteboard to demonstrate how to create a simple program that includes a cause-and-effect relationship using Game Changineer. Start by declaring some objects into the game, such as:

     ```
     There is a rabbit. There is a cat. There is a carrot. There is a ball.
     ```

   - Write example commands and sentences on the board:

     ```
     When the rabbit touches a carrot, the carrot disappears.
     ```

```
When the cat touches the ball, it kicks the ball.

When the rabbit touches the cat, it stops.
```

Guided Practice (20 minutes):

1. **Hands-On Activity:**

- Have students go on to Game Changineer platform.
- Guide them through the process of adding characters. Example: “There are 10 carrots. There is a rabbit.”
- Review: Guide the students to write sentences that describe how the rabbit moves with arrow keys.
- Encourage students to write sentences describing the cause-and-effect relationships they are programming.

2. **Monitoring and Support:**

- Walk around the classroom to provide assistance and answer questions.
- Check that students are correctly writing the commands and corresponding sentences.

Independent Practice (10 minutes):

1. **Challenge:**
   - Ask students to add 5 foxes to the game.
   - Ask students what cause-effect relationships exist between the fox and rabbit. Example:

```
When a fox sees the rabbit, it chases the rabbit.

When a fox touches the rabbit, the rabbit dies.
```

   - Have them write sentences to describe these new relationships.
   - Now consider the following cause-effect sentence:

```
When a fox sees the rabbit, it chases it.
```

   - Enter the above sentence into Game Changineer. Click Execute and see the yellow tags on the left panel. What problems exist for this sentence? How should the sentence be fixed?
   - Hint: ‘it chases it’ should be ‘it chases the rabbit’.
   - Note: the generated code was lucky to have interpreted the two pronouns correctly in the original ‘it chases it’. However, we should replace the phrase with the clear version, so that there would not be room for misinterpretation.

Closure (5 minutes):

1. **Review:**

    - Recap the key points of the lesson: cause and effect in programming and writing descriptive sentences.
    - Ask a few students to share their sentences and demonstrate their projects.

2. **Homework (Optional):**

    - Assign a homework in which there are bones, a puppy and several coyotes.
    - The students are to design a game involving these characters. Make sure the cause-effect relationships are described correctly.

Assessment:

- Observe students during the hands-on activity to ensure they understand the concepts.
- Review the sentences students write to check for accuracy and clarity.

Helpful Points to Watch For:

- Ensure students understand the relationship between the cause (antecedent event) and the effect (consequent action).
- Encourage students to use clear and descriptive language in their sentences.
- Provide support for students who may struggle with the programming environment or writing sentences.

Lesson Plan 3: Maze

Subject: STEM / Language Arts / Computer Science / Computational Thinking
Duration: 60 minutes

Learning Objectives:

- Students will learn how to place characters using a text-based map.
- They will understand how the 600×600 pixel canvas divided into 20×20 blocks.
- Understand the GameChangineer map syntax.
- Learn how the map translates to visual placement on the canvas.
- Practice modifying the map to create a maze challenge.

Introduction

In almost all of the games, the starting positions of all the characters are fixed. Think about the games that you like to play. Are the characters placed at the same positions every time you play the game?

This is achieved by the concept of a map. Specifically, in video games, this is called a 'tilemap'. A sample starter map is shown below:

```
---r--------------
------------------
-----w---w--------
------------------
----------c-------
------------------
----w-------w-----
------------------
------------------
```

- Copy the above map into the **map** textbox (smaller textbox below the GamePlan textbox) on the right panel of the GameChangengineer website.
- Click the "Execute" button.
    - Observe the generated canvas: The canvas is 600×600 pixels, divided into 20×20 blocks.
    - The rabbit (`r`) appears near the top.
    - A carrot (`c`) and walls (`w`) are placed as indicated.

Note that each row should have exactly 20 characters to align with the grid.

Explanation of the Map Syntax:

Map Structure: Canvas Grid: The canvas is divided into 20 columns by 20 rows (here, we use 10 rows for a simplified example).

Characters:

- `r` stands for the rabbit.

- `c` stands for the carrot.
- `w` stands for a wall (obstacle).
- `-` stands for a blank space.

- **How It Works:** The letters in each row determine the positions on the grid. For example, the rabbit is placed by the character `r` in the second row. Walls (`w`) create obstacles, and carrots (`c`) are placed for the rabbit to collect. Click on the red **Show/Hide Map Keys** to see the individual keys corresponding to each available character

Problem-Based Challenge

Challenge Description:

Task: Modify the earlier provided map to create a maze in which two carrots are placed, and there should exist a path from the rabbit to the carrots.

Requirements: Add a second carrot: Place a new carrot (`c`) in a different area.

Rearrange walls: Adjust the placement of walls (`w`) to form a maze-like path that requires the rabbit to take a non-direct route to reach the carrots.

Your Challenge:

- Create a map that meets these goals.
- Ensure every row has exactly 20 characters.
- Make sure that the maze is solvable (i.e., open paths exist).

Hints with a Limit (Students may request up to three hints)

- **Hint 1:** Verify that each row in your map has at most 20 characters. Consistency is key for the grid to work correctly.
- **Hint 2:** Think about the path the rabbit must follow. Use walls (`w`) to block direct paths, but ensure there are enough dashes (`-`) to form a navigable maze.
- **Hint 3:** Place the second carrot (`c`) in an area that is not immediately accessible from the starting position of the rabbit, encouraging the creation of a maze.

Immediate Feedback

Running Your Map: When you click "Execute," the tool will convert your text map into a game stage.

What to Look For: The rabbit appears in its designated starting position.

Two carrots appear at your specified locations.

Walls appear as obstacles forming a maze.

If there are syntax errors (e.g., rows with the wrong number of characters), the tool will prompt you with error messages.

Outcome: Successful execution and a visual stage that reflects your intended maze design means you've met the challenge requirements.

Notes

Map Syntax Details: Canvas Size: The canvas is 600×600 pixels.

Grid Division: The canvas is divided into 20 columns (each represented by a character) per row.

Character Keys:

- r: Rabbit
- c: Carrot
- w: Wall
- -: Blank space

- **Whitespace vs. Dash:** Remember that the dash (-) explicitly marks a blank grid cell. (Review the quiz on the differences if needed.)

- **Programming Concepts:** This lesson reinforces ideas like spatial mapping, grid-based positioning, and how simple text can control complex visual outputs.

Sample Correct Answer

Below is one example of a correct solution to the challenge.

Sample Map Answer:

```
--------------------
----r---------------
-----w-w-w-w--------
-----w---w----------
-----w---w-----c----
-----w---w----------
-----w---w-w-w------
-----w-----------c--
--------------------
--------------------
```

Explanation of the Example:

Grid Consistency: Each row contains exactly 20 characters.

Character Placement: The rabbit (r) is in the second row.

Two carrots (c) are placed in rows 5 and 8.

Walls (w) are arranged to form a maze-like barrier in rows 3–8.

Maze Design: The arrangement of walls forces the rabbit to follow a longer, indirect route to reach the carrots, fulfilling the challenge's goal.

Progression

After successfully modifying the map:

Further Exploration: Experiment by adding additional obstacles (e.g., place a fox (x) and other characters).

Advanced Challenge: Use the “Execute” feature to view the impact of your changes in real time and refine your maze design for optimal gameplay.

Summary

This lesson plan guides you through using GameChangineer’s map syntax to create a game stage. You start with provided maze, learn by observing how the map translates to a game, and then modify the map to meet specific challenges. The correct answer example shows one way to meet the challenge, but there are many valid solutions as long as the syntax is followed and the maze is functional. Happy mapping!

Lesson Plan 4: Bullets

Subject: STEM / Language Arts / Computer Science / Computational Thinking
Duration: 60 minutes

Learning Objectives:

- Students will understand how to create events that trigger shooting actions
- Students will learn interactions between characters and bullets
- Students will practice writing sentences that describe these actions.

Introduction (10 minutes):

1. **Hook:** Start with a brief discussion about action games and how characters can shoot bullets. Ask students if they have ever played a game where they control shooting.
2. **Objective:** Explain that today they will learn how to program a character to shoot bullets and write sentences to describe these actions.

Direct Instruction (15 minutes):

1. **Concept Explanation:**

   - Review the concept of events in programming and how they can trigger actions, as in “When [event], then [action]”
   - Show examples of sentences that describe these actions, such as "When the space bar is pressed, the character shoots a bullet."
   - Show a second example: “When the fox sees the rabbit, it shoots at the rabbit.”

2. **Demonstration:**

   - Use a projector or interactive whiteboard to demonstrate how to create a simple program where a character shoots bullets.
   - Write example commands and sentences on the board:

```
There are 20 diamonds and a rabbit.

When the space bar is pressed, the rabbit shoots.

When a diamond is shot, it explodes.
```

Guided Practice (20 minutes):

1. **Hands-On Activity:**

   - Have students go on to Game Changineer platform.

- Review: Guide them through the process of setting up the characters in the game (could be through the map textbox)
- Show them how to set up events that trigger the shooting action.
- Show them how to describe interactions with the bullet: "When a diamond is shot, …"
- Encourage students to write sentences describing these actions.

2. **Monitoring and Support:**

- Walk around the classroom to provide assistance and answer questions.
- Check that students are correctly writing the commands and corresponding sentences.

Independent Practice (10 minutes):

1. **Challenge:**
   - Ask students to create a new game using their own chosen characters.
   - Have them write sentences to describe shooting and interactions.
2. **Challenge 2:**
   - The characters in a game are divided into the player character (PC) and non-player characters (NPC)
   - The player character (PC) is the character that the user controls. The PC can always be shot.
   - Not all NPCs should be shot (in order to prevent violent games). Identify 3 different NPC character types (such as the rabbit, etc.) that should not be shot in a game.

Closure (5 minutes):

1. **Review:**

- Recap the key points of the lesson: programming events to shoot and writing descriptive sentences.
- Ask a few students to share their sentences and demonstrate their projects.

Assessment:

- Observe students during the hands-on activity to ensure they understand the concepts.
- Review the sentences students write to check for accuracy and clarity.

Helpful Points to Watch For:

- Ensure students understand how characters can shoot.
- Encourage students to use clear and descriptive language in their sentences.
- Provide support for students who may struggle with the programming environment or writing sentences.

Lesson Plan 5: Jumping

Subject: STEM / Language Arts / Computer Science / Computational Thinking
Duration: 60 minutes

Objectives:

- Students will understand how to create events that trigger jumping actions.
- Students will learn the specifics of describing how a character jumps.
- Students will vary jumping height, landing, double jump, etc.

Introduction (10 minutes):

1. **Hook:** Start with a brief discussion about characters in games that can jump. Ask students if they have ever played a game where they control a character's jumping.
2. **Objective:** Explain that today they will learn how to program a character to jump, double jump, landing on objects, and write sentences to describe these actions.

Direct Instruction (15 minutes):

1. **Concept Explanation:**

    - Review the concept of events in programming and how they can trigger actions: When [event], then [action]. Such as “When a fox sees a rabbit, it chases the rabbit. When spacebar is pressed, the rabbit jumps.”
    - Explain the basic idea of creating a jumping action for a character. The height of the jump is directly related to the speed of the character. But this can be modified, as explained below.
    - Show examples of sentences that describe these actions, such as "When the space bar is pressed, the rabbit jumps 9 pixels."
    - Certain objects, such as the brick, are automatically landable. If you wish to make brick not landable, simply state “The brick is not landable.”
    - Most of the objects are not landable by default. To enable landing, there are two methods:
        - `The cheese is landable.`
        - `When the rabbit lands on a cheese, the rabbit stops.`

2. **Demonstration:**

    - Use a projector or interactive whiteboard to demonstrate how to create a simple program where a character jumps.
    - Whenever a character is able to jump, then the game is said to be a 2.5D game (rather than a simple 2D game). This is not a full 3D game, since no 3D graphics are used.
    - Write example commands and sentences on the board:
        - `There are 20 bricks near the bottom.`
        - `When the space bar is pressed, the rabbit jumps.`

Guided Practice (20 minutes):

1. **Hands-On Activity:**

   - Have students go on to the GameChangineer platform.
   - Review: Guide them through the process of adding characters (could be through the map textbox).
   - Make sure there is only one player character, such as the rabbit, that is controlled by the user.
   - Show them how to set up events that trigger the jumping action.
   - Show them how to land on the objects.
   - Encourage students to write sentences describing these actions.

2. **Monitoring and Support:**

   - Walk around the classroom to provide assistance and answer questions.
   - Check that students are correctly writing the commands and corresponding sentences.

Independent Practice (10 minutes):

1. **Challenge:**
   - Ask students to change the jumping height: When spacebar is pressed, the rabbit jumps 15 pixels.
   - Killing enemies: When the rabbit lands on a fox, the fox explodes.
2. **Challenge 2 – double jump:**
   - Write the following sentence: The rabbit is aerial.
   - Observe what happens when spacebar is pressed before the rabbit lands.

Closure (5 minutes):

1. **Review:**

   - Recap the key points of the lesson: programming events to make a character jump and writing descriptive sentences.
   - Recap: landing and double jump. How to make an object landable? How to enable a character to double jump?
   - Ask a few students to share their sentences and demonstrate their projects.

2. **Homework (Optional):**

   - Assign students to design a game with platforms placed in the map textbox, and the player character needs to jump and collect items.

Assessment:

- Observe students during the hands-on activity to ensure they understand the concepts.
- Review the sentences students write to check for accuracy and clarity.

Helpful Points to Watch For:

- Ensure students understand how to create and control jumping actions for characters.
- Ensure students are able to explain landing and height of jumps.
- Encourage students to use clear and descriptive language in their sentences.
- Provide support for students who may struggle with the programming environment or writing sentences.

Lesson Plan 6: Abstraction

Subject: STEM / Language Arts / Computer Science / Computational Thinking
Duration: 60 minutes

Objectives:

- Students will understand how to identify opportunities to apply abstraction concepts.
- Students will learn the specifics of abstracting a description.

Introduction (10 minutes):

1. **Hook:** Start with a brief discussion about what it means to be verbose in description; that is, description containing unnecessary details.
2. **Objective:** Explain that today they will learn how to remove unnecessary details and focus only on the primary reasons for describing a scenario.

Direct Instruction (15 minutes):

1. **Concept Explanation:**

   - Review the concept of events in programming and how they can trigger actions: When [event], then [action]. Such as "When a fox sees a rabbit, it chases the rabbit. When spacebar is pressed, the rabbit jumps."
   - Explain the basic steps of identifying unnecessary details. Key Step: ask yourself if all the details are necessary and can we remove any detail and still maintain the key idea in the description.
   - Show examples of sentences that describe these actions, such as "*When I watch TV and do not finish my homework, I will not receive full grade on the assignment*." This can be abstracted to simply "*When I do not finish my homework, I will not receive full grade on the assignment*." The first antecedent, "*watch TV*", is unnecessary detail and can be removed without affecting the resulting consequent.

2. **Demonstration:**

   - Use a projector or interactive whiteboard to demonstrate how to create a simple program where a character jumps.
   - Write example game plan and sentences on the board:
     - `There are 20 carrots.`
     - `There are 5 foxes and a rabbit.`
     - `When an arrow is pressed, the rabbit moves in the same direction.`
     - `When the rabbit touches a carrot, the carrot disappears.`
     - `When a fox touches the rabbit, game over.`
     - `When a fox does not touch the rabbit and all carrots are gone, you win.`

- Determine which detail is unnecessary in the last sentence above. Would "*When all carrots are gone, you win.*" be sufficient? Is the first condition ("*fox does not touch the rabbit*") unnecessary?

Guided Practice (20 minutes):

1. **Hands-On Activity:**

- Have students go on to the GameChangineer platform.
- Review: Guide them through the process of adding characters (could be through the map textbox).
- Make sure there is only one player character, such as the rabbit, that is controlled by the user.
- Show them how to identify verbose sentences that contain unnecessary detail. Show them how to focus on the main idea.
- Encourage students to write sentences describing these actions.

2. **Monitoring and Support:**

- Walk around the classroom to provide assistance and answer questions.
- Check that students are correctly writing the sentences.

Independent Practice (10 minutes):

1. **Challenge:**
   - Ask students to add jumping to the game (may need to review previous lesson on jumping).
   - Ask one group of students to write complex sentences that contain unnecessary details and have other students identify the necessary portions and remove the unnecessary details.

Closure (5 minutes):

1. **Review:**

- Recap the key points of the lesson: programming events to identify unnecessary details.
- Recap: How to identify unnecessary details? What questions do you ask yourself to determine if a detail can be removed?
- Ask a few students to share their sentences and demonstrate their projects.

2. **Homework (Optional):**

- Assign students to design sentences with and without unnecessary details. Compare and contrast these sentences.

Assessment:

- Observe students during the hands-on activity to ensure they understand the concepts.
- Review the sentences students write to check for accuracy and clarity.

Helpful Points to Watch For:

- Ensure students understand how to identify unnecessary details.
- Ensure students are able to explain how they found unnecessary details. They are thus "abstracting" their ideas to focus only on the necessary details.
- Encourage students to use clear and descriptive language in their sentences.
- Provide support for students who may struggle with the programming environment or writing sentences.