Chapter 6: Conditional Statements

6.0.1 Objectives

Students typically find conditional statements (also known as selection, or decision statements) easy to understand when compared to other constructs. The main area of confusion involved with conditionals occurs when students begin using Boolean combinations: for example, the word "and" as it Is used casually in English can have a different for the formal word "and" in Boolean logic. At the end of this unit, students should be able to apply conditionals, creating Boolean expressions with comparisons, and they should be able to reason correctly about the use of the Boolean operators "and" "or" and "not."

6.0.2 Topic Outline

6.0 Chapter Introduction
- 6.0.1 Objectives
- 6.0.2 Topic Outlines
- 6.0.3 Key Terms
- 6.0.4 Key Concepts

6.1 Lesson Plans
- 6.1.1 Suggested Timeline
- 6.1.2 CSTA Standards
- 6.1.3 Lesson Plan I on using the control block- If, If… Else.
- 6.1.4 Lesson Plan II on re-visiting the spiral assignment on the Question Bot.
- 6.1.5 Lesson Plan III on using Boolean expressions.
- 6.1.6 Lesson Plan IV to designing a hi-lo game and the race car game.

6.2 Resources
- 6.2.1 Additional exercises.

6.0.3 Key Terms

Boolean Values: true / false	If then else, If then else if else
Simple & Complex expressions	`match` functionality
AND / OR – Operators	
`is` (comparison), `isnt`, < ,>	
Numerical comparisons	
String comparisons	

6.0.4 Key Concepts

Controlling Code Using Conditions

The word `if` can be used to put a block of code under the control of a condition, so it only runs when the condition is true.

Below is an example that uses `if` to control turtle motion by testing keyboard presses.

The indented code `fd 2` only runs when the condition `pressed('W')` is true, that is, when the user is pressing the W key. Similarly, the two lines of code `rt 2; dot blue, 5` only run when the user is pressing the D key.

If neither key is pressed, neither block of indented code is run. If both keys are pressed, both blocks are run.

```
forever ->
  if pressed('W')
    fd 2
  if pressed('D')
    rt 2
    dot blue, 5
```

Testing key presses using if.

Using “else” For the Other Alternative

The “else” keyword allows you to program a second action to take when the “if” does not happen. The second block of code will run when the condition is false.

```
forever ->
 if pressed('W')
   fd 2
 else
   rt 2
```

Providing two alternatives using if/else.

This program moves the turtle forward when W is pressed. When W is not pressed, it spins the turtle.

Chaining “else if” for Multiple Alternatives

When there are three or more actions, `if` and `else` can be chained.

```
forever ->
 if pressed('W')
   fd 2
 else if pressed('S')
   bk 2
 else
   rt 2
```

Chaining if, else if, and else for three alternatives.

This code moves forward if W is pressed and backward if S is pressed. It spins right if nothing is pressed. A chained if/else only chooses the first condition that is true, so if both W and S are pressed at the same time, this program will just do “fd 2” and not move backward.

Using “and”, “or” and “not” to Combine Boolean Expressions

The words `and`, `or`, and `not` are **Boolean operators** that can be used to combine conditions. For example, the following program uses “and” and “not”. It draws a blue ring and then only moves the turtle forward if W is pressed and the turtle is not already touching the blue ring.

```
dot blue, 500
dot white, 400
forever ->
 if pressed('W') and not touches('blue')
   fd 2
 else
   rt 2
```

Combining tests using and and not.

Although Boolean operators usually work in the same way as when reading them as English words, it is important to understand exactly how they work as mathematical operators because it is easy to get unexpected effects.

Confusing “and” With “or”

Consider a program that where “up” and “W” keys need to work equivalently, both working in the same way to move the turtle forward. We might be tempted to use the “and” combiner to capture both cases with a single “if” like this:

WRONG:

```
forever ->
  if pressed('up') and pressed('W')
    fd 2
```

Incorrectly using “and” to combine two alternatives.

This code, however, will not generate the desired effect! To understand why, we need to understand how `and` and `or` operate on truth values.

Boolean Values and Boolean Tables

The words `and`, `or` and `not` are Boolean operations that combine “true” and “false” values (similar to the arithmetic rules you get when using “+”, “*” to combine regular numbers). Just as we can learn about addition and multiplication by creating addition and multiplication tables, we can understand `and` and `or` by writing **truth tables**. Here are two truth tables related to the program above:

pressed('up') **and** pressed('W')	pressed 'up' = false	pressed 'up' = true
pressed 'W' = false	false	false
pressed 'W' = true	false	true

pressed('up') **or** pressed('W')	pressed 'up' = false	pressed 'up' = true
pressed 'W' = false	false	true
pressed 'W' = true	true	true

The conjunction `and` combines two Boolean values and creates "true" only when both of the values are true. For example, `pressed('up') and pressed('W')` is true only when both the up and W keys are pressed at the same time.

The disjunction `or` combines two Boolean values and creates "true" when either or both of the values are true. For example, `pressed('up') or pressed('W')` is true when just the W key is pressed, or just the up key is pressed, or both. This is what we want for our program.

To fix the program, the `and` should be switched to `or`.

Testing Numbers Using Comparison Operators

Boolean expressions can be used to test the properties of numbers. Most of the comparison operators you would see in math class work in a programming language, but they may be written with slightly different punctuation. For example, "is less than or equal to" is written `<=`. Here is a summary of some common Boolean tests for numbers:

Expression	Description	What if x = 0?	What if x = 3?	What if x = 6?
`x is 3`	x is equal to 3	false	true	false
`x isnt 3`	x is not equal to 3	true	false	true
`x < 3`	x is less than 3.	true	false	false
`x <= 3`	x is less than or equal to 3	true	true	false
`x > 3`	x is greater than 3	false	false	true
`x >= 3`	x is greater than or equal to 3	false	true	true
`0 < x <= 6`	x is greater than 0 and less than or equal to 6	false	true	true
`x % 2 is 1`	x is odd (because it has remainder 1 when divided by 2)	false	true	false
`x % 3 is 0`	x is divisible by 3	false	true	true

Confusing "or" with Comparisons

Numerical comparisons can be combined with Boolean operators. For example, `(x > 6 and x isnt 9)` means that x is a number greater than 6 other than 9, and `(x is 5 or x is 11)` means that x is either 5 or 11. It is important, though, to remember that the word "`or`" operates on truth values and not on numbers, so the version of the program on the left does not do produce the desired result.

WRONG:	RIGHT:

`await readnum 'How many items?', defer n`
`if n is 1 or 2`
` write 'Come to the speedy checkout.'`
	`await readnum 'How many items?', defer n`
	`if n is 1 or n is 2`
	` write 'Come to the speedy checkout.'`

The program on the left incorrectly results in the speedy checkout line no matter what number you enter.

To understand why, remember that `or` operates on truth values, so when you say “`or 2`”, it begins with the question “is 2 true or false?” By convention, any number that is not zero is treated as “true”, so “`or 2`” makes the expression always true regardless of the value of num. On the other hand, the program on the right produced the desired result: “`or n is 2`” only makes the expression true when the number is 2.

Another way to think about the difference is with precedence of operators. The word “`or`” has lower precedence than the word “`is`”, so the expression on the left reads like this: `((n is 1) or 2)` and the expression on the right reads like this: `((n is 1) or (n is 2))`.

Testing Strings Using Pattern Matching

Text strings can also be tested to create Boolean values. It is common to test strings by comparing them exactly (looking at their length) or by testing if the string matches a pattern using the “match” method. Pattern matching can be used to determine if a string contains a particular pattern of letters within it.

The following table shows several examples.

Expression	Description	"appear"	"pear"	"peachy"
`x is "pear"`	x is exactly equal to the string "pear"	false	true	false
`x.length is 6`	x has exactly 6 characters	true	false	true
`x.match(/pp/)`	x contains the substring “pp”.	true	false	false
`x.match(/pea/)`	x contains the substring “pea”	true	true	true
`x.match(/PEA/)`	x contains the substring “PEA”	false	false	false
`x.match(/Pea/i)`	x contains the substring “Pea”, ignoring case	true	true	true
`x.match(/^pea/)`	x contains “pea” at the start of the string	false	true	true
`x.match(/ear$/)`	x contains “ear” at the end of the string	true	true	false
`x.match(/a(p|ch)/)`	x contains “a” followed by either “p” or “ch”	true	false	true
`x.match(/ap*e/)`	x contains “a”, then zero or more “p”, then “e”	true	false	false

The patterns used between the “/” symbols are called **regular expressions**.

A regular expression can be used to test whether a string contains a fixed pattern, for example whether it contains the letters “pp”. Normally regular expressions are case-sensitive, so “PEA” does not match “pea”, but putting an “i” after the regular expression makes it case-insensitive.

Regular expression patterns have several powerful features. For example, in a regular expression, “^” matches the beginning of the string, “$” matches the end of the string, “(one|other)” is used to match alternatives, and “*” allows a sub-pattern to be repeated zero or more times.

Although the types of patterns shown above are enough for most situations, regular expressions have several more features. There are many excellent resources about regular expressions on the Internet if you search for “regular expression lessons”. When exploring, it is important to know that the symbols used in regular expression patterns are standardized, and the same pattern language is used in JavaScript, CoffeeScript, Python, Perl, Java, C# and other languages.

6.1.1 Suggested Timeline: 1 55-minute class period

Instructional Day	Topic
2 Days	Lesson Plan I: If, If Then Else statements using fun visual elements
1 Day	Lesson Plan II: Question Bot & Lesson Plan III Complex If Statements
1 Day	Lesson Plan IV: Pair programming for the HiLo Game
1 Day	Lesson Plan V: Race Car Track game

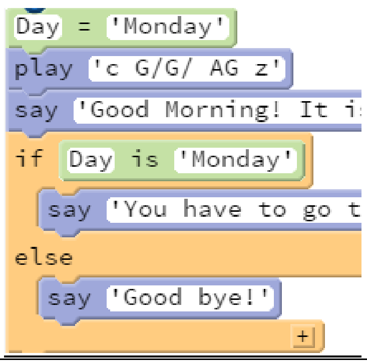
6.1.2 Standards

CSTA Standards	CSTA Strand	CSTA Learning Objectives Covered
Level 3 A (Grades 9 – 12)	Computational Thinking (CT)	Explain how sequencing, selection, iteration and recursion are building blocks of algorithms.
Level 3 A (Grades 9 – 12)	CT	Explain the program execution process.
Level 3 A (Grades 9 – 12)	CT	Describe how mathematical and statistical functions, sets, and logic are used in computation.

6.1.3 Lesson Plan I

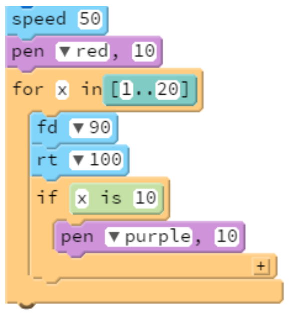
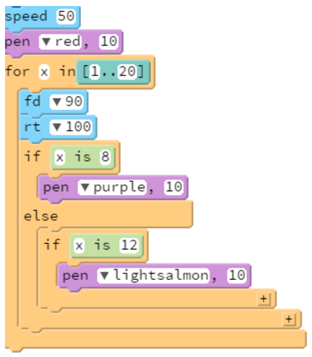
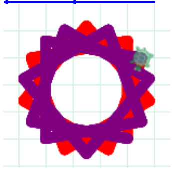
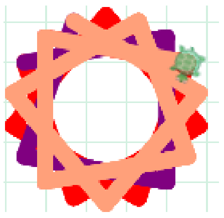
This lesson focuses on the Control Block, If statements and If Else statements. The lesson should take about 30 to 40 minutes of a class period, providing time for students to code programs similar to the ones modeled.

Content details	Teaching Suggestions	Time

Code
`x = random [1..3]`
`write x`
`if x is 1 or x is 2`
`  write 'Today is your lucky day!'`
`else`
` if `` is `` `
`    write 'Stay low. Let everything happen tomorrow'`
`  else`
`    write 'I cannot see your future.'`
` # Demonstrate complex IF Statements`

`Day = 'Monday'`
`play 'c G/G/ AG z'`
`say 'Good Morning! It i`
`if Day is 'Monday'`
`  say 'You have to go t`
`else`
`  say 'Good bye!'`
	Explain the key concept of the evaluation of a Boolean expression.
	
	Show the Control Block and the 'IF' statement.
	
	Type the code shown on the left column. (Note: You will need speakers for this program. You can use the `write` block instead of `say` block.)
	
	Teaching Tips: You can extend this lesson further by adding a loop around the entire code for the tune to be played for a fixed period (e.g. 5 times). Explain that the program gives an output based on the value the variable to which "Day" is set.
		Demonstration: 20 minutes

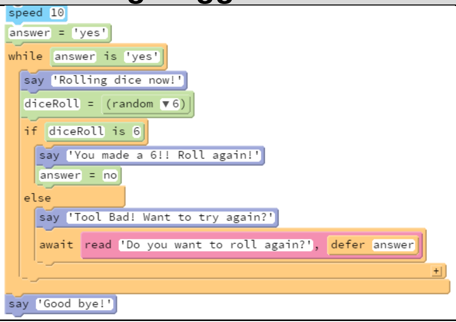
Code	Give another example using patterns. Demonstrate how the conditional can impact	Demonstration: 20 minutes

Content details	Teaching Suggestions	Time

`speed 50`
`pen red, 10`
`for x in[1..20]`
  `fd 90`
  `rt 100`
  `if x is 8`
    `pen purple, 10`
  `else`
    `if x is 12`
      `pen lightsalmon, 10`
 	

`speed 50`
`pen red, 10`
`for x in [1..20]`
  `fd 90`
  `rt 100`
  `if x is 10`
    `pen purple, 10`
 	

`speed 50`
`pen red, 10`
`for x in [1..20]`
  `fd 90`
  `rt 100`
  `if x is 8`
    `pen purple, 10`
  `else`
    `if x is 12`
      `pen lightsalmon, 10`
	the color in which the pattern is drawn.
	
	Teaching Tips: Change the values within the conditional to show how the pattern changes.
	
	Add another if statement to show another color. (Point out the use of a nested if statement.)
	
	Add an 'if' statement to change the speed. Here is the copy of the program: http://teachersguide.pencilcode.net./edit/chapter6/pattern
	
	
	Now students can start writing their own programs. Students are expected to write both the programs that were demonstrated. Students should complete both programs by the end of the class period (15 minutes of class time).
	
	
		Student Practice: 15 minutes

Code
`speed 10`
`answer = 'yes'`
`hide()`
`while answer is 'yes'`
  `diceRoll = (random 6)`
  `label`
`String.fromCharCode(9856 +`
`diceRoll), 100`
  `say 'Rolling dice now!'`
  `if diceRoll is 6`
    `say 'You made a 6!! Roll`
`again!'`
    `answer = no`
  `else`
    `write diceRoll`
    `say 'Tool Bad! Want to`
	Add a few fun elements to the program to show that how using a loop and a conditional increases a program's power.
	
	Here is the code to simulate the roll of a die. (This is a starter program for a Yahtzee game. The Additional Exercises section provides the specifications for Yahtzee.)
	
	Demonstrate and walk students through the code. Point out the use of random numbers, how the assignment of an exit condition lets the loop exit eventually, and the If… Else block.
		Demonstration: 55 minutes

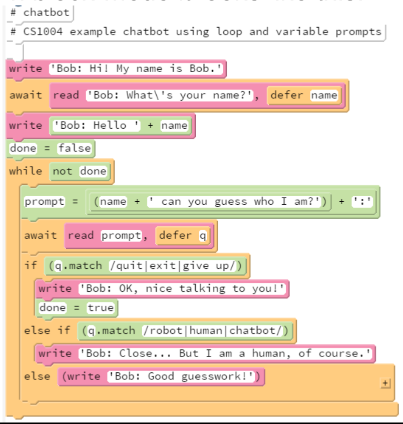
Content details	Teaching Suggestions	Time

`try again?'`
`    await read 'Roll`
`again?', defer answer`
`say 'Good bye!'`
	
	`speed 10`
	`answer = 'yes'`
	`while answer is 'yes'`
	`  say 'Rolling dice now!'`
	`  diceRoll = (random ▼6)`
	`  if diceRoll is 6`
	`    say 'You made a 6!! Roll again!'`
	`    answer = no`
	`  else`
	`    say 'Tool Bad! Want to try again?'`
	`    await read 'Do you want to roll again?', defer answer`
	`say 'Good bye!'`	

6.1.4 Lesson Plan II

This lesson revisits the Spiral assignment Question Bot introduced in the Chapters 3, 4, and 5. Students will extend this program using all of the previously-learned concepts and add conditional statements.

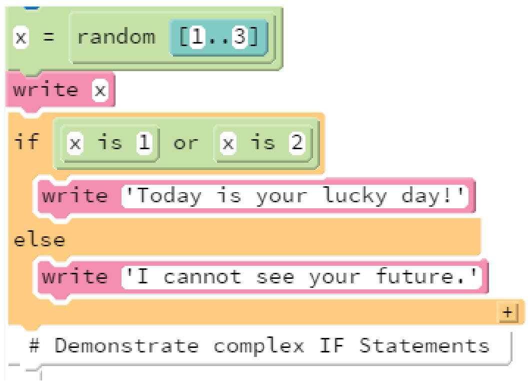
Content details	Teaching Suggestions	Time

Code:
`# chatbot`
`# CS1004 example chatbot using`
`loop and variable prompts`
 	
`write 'Bob: Hi! My name is Bob.'`
`await read 'Bob: What\'s your`
`name?', defer name`
`write 'Bob: Hello ' + name`
`done = false`
`while not done`
`  prompt = name + ' can you`
`guess who I am?' + ':'`
`  await read prompt, defer q`
`  if (q.match /quit|give up/)`
`    write 'Bob: OK, nice talking`
`to you!'`
`    done = true`
`  else if (q.match /bot/)`
`    write 'Bob: Close... But I`
`am a human, of course.'`
`  else`
`    write 'Bob: Good guesswork!'`
	Pull up the Question Bot (Loops version) program. Ask students how its functionality can be expanded.
	
	Even though the program asks four people's names, it does not "remember" them?
	
	Making the lesson more interesting by asking students for a good question. Walk through the code and emphasize the power of conditionals.
	
	This example also introduces the operator `x.match` () functionality. In block-mode it looks like this:
	
	`# chatbot`
	`# CS1004 example chatbot using loop and variable prompts`
	
	`write 'Bob: Hi! My name is Bob.'`
	`await read 'Bob: What\'s your name?', defer name`
	`write 'Bob: Hello ' + name`
	`done = false`
	`while not done`
	`  prompt = (name + ' can you guess who I am?') + ':'`
	`  await read prompt, defer q`
	`  if (q.match /quit|exit|give up/)`
	`    write 'Bob: OK, nice talking to you!'`
	`    done = true`
	`  else if (q.match /robot|human|chatbot/)`
	`    write 'Bob: Close... But I am a human, of course.'`
	`  else (write 'Bob: Good guesswork!')`
		Demonstration: 20 minutes

6.1.5 Lesson Plan III

This lesson plan demonstrates the use of complex if statements, specifically, combining multiple Boolean expressions.

Content details	Teaching Suggestions	Time

Code
`x = random [1..3]`
`write x`
`if x is 1 or x is 2`
`  write 'Today is your lucky day!'`
`else`
`  write 'I cannot see your future.'`
` # Demonstrate complex IF Statements`

`x = random [1..3]`
`write x`
`if x is 1 or x is 2`
`  write 'Today is your lucky day!'`
`else`
`  write 'I cannot see your future.'`
`+`
`# Demonstrate complex IF Statements`
	Explain the key concept of the evaluation of multiple Boolean expressions. Explain how an AND / OR operator combines two expressions and evaluates them.
	
	The example provided here is a simple program that acts like a "Genie" and predicts the day based on the random number generated.
	
	Teaching Tip: Explain how an OR operator works. Explain using the example that the 'if' statement will get executed even if one of the expressions is true.
	
	Ask the students to copy the code provided and test the program.
		Demonstration: 20 minutes

Code:
`x = random [1..3]`
`write x`
`if x is 1 or x is 2`
`  write 'Today is your lucky day!'`
`else`
` if `` is ```
`    write 'Stay low. Let everything happen tomorrow'`
`  else`
`    write 'I cannot see your future.'`
` # Demonstrate complex IF Statements`
	Now explain in-depth using the modified program as show in the left column.
	
	Teaching Tip: Explain how an AND operator works. Explain using the example that the 'if' statement will get executed IF AND ONLY IF both expressions are true.
	
	Ask students to copy the code provided and test the program.
	
	Give students 10 minutes to experiment with if statements and modifications of the code. Let them test other programmers' code and see what kind of day the genie predicts for them!
		Demonstration: 20 minutes.
		
		Student Practice: 10 minutes

6.1.6 Lesson Plan IV

Students will now work with a partner to design a Hi-Lo Guessing game. The Additional Exercises section provides the problem description. Students will watch a short video on pair programming before beginning the exercise. The lesson plan below focuses on how to design a project with a partner. Students will take about half a class period to design the program. They will have the rest of the class period and one more class period or homework to complete the assignment. (Note: Homework may not result in true collaborative work because it increases the temptation to find answers on the Internet and from people outside of the team.)

Content details	Teaching Suggestions	Time
https://www.youtube.com/watch?v=vgkahOzFH2Q	Have students watch the video on pair programming.	Student Practice: 5 minutes
Split students into groups. The References section provides resources on forming successful collaborative groups. Using the “Rally Robin” co-operative learning® structure, have the students write the pseudocode for the HiLo game. Their pseudocode should indicate: i. Variables used ii. Control structures needed iii. Input / Output statements Student Practice: 20 minutes		
Rally Robin instructions: On a piece of paper, take turns writing instructions on how to design the guessing game. Each instruction should be a numerical bullet (e.g. 1,2,…) Student 1 writes the first instruction. Student 2 writes the second instruction. Keep repeating this process until all of the instructions have been recorded. Next, go back and take turns revising the instructions until both partners are satisfied. Submit your work your teacher for grading. Here is a sample Hi-Lo program. Student Practice: 20 minutes		
On your computer, type the program with your partner using the pair programming methods presented in the pair programming video. Student Practice: 40 minutes		

6.1.7 Lesson Plan V

This interactive lesson involves having the students design a racecar game. We recommend that students type this in text using CoffeeScript. Encourage the students to toggle between text-mode and block-mode to venture out of their comfort zone and increase their confidence. Follow the activity Turtle Race track as provided in this activity sheet

Content details	Teaching Suggestions	Time
http://activity.pencilcode.net/home/worksheet/race.html	Teaching Tip: Have all the students in the class start on the worksheet at the same time. Print out the worksheet and give it to the students. This will eliminate the temptation to copy-paste the code. Follow the link to complete steps 1 and 2. Instruct students to type the text and stay in text-mode.	Student Practice: 10 mins
http://activity.pencilcode.net/home/worksheet/race.html	Ask students to answer Questions 1 and 2 verbally in the classroom.	Student Practice: 15 mins
http://activity.pencilcode.net/home/worksheet/race.html	Now let the students venture out at their own pace as they respond to Challenges 1 through 4. Students can stay in text-mode or switch back to block-mode. If they do switch to block-mode, encourage them to toggle to text and see how their code looks. The solution code is given in CoffeeScript. Teaching Tip: If the students type the text, encourage them to indent their code to convey intent. All good programming practices (Refer Appendix A) should be followed.	Student Practice: 20 mins

The race car game activity can be completed by students.

1. http://activity.pencilcode.net/home/worksheet/race.html - the basic game
2. http://activity.pencilcode.net/home/worksheet/race-car.html - making the car look like a car, and this emphasizes the idea of an "object" whose behavior your control
3. http://activity.pencilcode.net/home/worksheet/race-two.html - here you introduce a "second object" - two instances - and now you can controls them separately
4. http://activity.pencilcode.net/home/worksheet/race-track.html - this is a review of drawing, but used for a very different purpose - to create a track shape
5. http://activity.pencilcode.net/home/worksheet/race-speed.html - this is an introduction to variables, used to keep track of how fast each car goes.
6. http://activity.pencilcode.net/home/worksheet/race-time.html - this is another use of variables, this time to keep track of how much time has passed
7. http://activity.pencilcode.net/home/worksheet/race-menu.html - this is an example of use of functions to divide your program into subprograms.

6.2 Resources

Additional Exercises:

Yahtzee
Design a modified version of Yahtzee where you roll three dice and the score is calculated based on which of the following categories are satisfied. The game ends after each user has had three turns. The user who has the largest point value at the end of three turns is the winner. The three categories are:

i. Three of a kind – 5 points
ii. Two of a kind – 10 points
iii. Yahtzee – You Win! Game over!

Hi-Lo- Guessing Game
Design a simple game where the computer generates a random number and the user must guess the number. The computer helps the user by giving responses such as “too high or too low”. The user has a fixed number of tries to guess the right answer. Once the allowed number of tries are completed, the game displays the computer-generated number and says “Game Over”. You can get up to extra five creativity points if you have added a new feature to the game.