



KAREL JR 3  
**STUDENT JOURNAL**

REVISED APRIL 19, 2017

NAME	
DATE STARTED	DATE COMPLETED
SCHOOL, CLASS, PERIOD	



## TABLE OF CONTENTS:

WELCOME TO YOUR JOURNAL	4
SECTION 11: USING THE KEYWORD DEF	5
SECTION 12: USING DEFINED COMMANDS; ADVANCED MAZE SKILLS	9
SECTION 13: COMPARING PROGRAMS; SOLVING COMPLEX PROBLEMS	13
SECTION 14: VARIABLES AND FUNCTIONS: INC(), DEC(), AND PRINT	17
SECTION 15: VARIABLES AND FUNCTIONS: (INC), DEC(),RETURN, LOCAL/GLOBAL	21
REVIEW YOUR PROGRESS	25
LIST OF BASIC COMMANDS AND KEYWORDS	26
LIST OF KEY VOCABULARY	28
FILE LOG: GAMES I HAVE CREATED	31
DESIGN TEMPLATE	32
NOTES	33

General Website: <https://nclab.com/>

Karel Gallery : <https://nclab.com/karel-gallery/>

Desktop (needs login information) <https://desktop.nclab.com/>

*Keep your name and password in a safe place.*

## WELCOME TO YOUR JOURNAL

**Welcome to Karel 3. You have learned how to tie instructions together based on their relationships. Is an action or set of actions repeated? You can use repeat loops. Will an action only be needed under certain conditions? You can use combinations of conditions and loops to control those actions. Now, you are ready to take your programming skills to the next level.**

**In Karel 3, you learn to define sets of commands to be used in a program. You are solving more complex problems, and choosing the best way to solve them. You are using functions such as `inc()` and `dec()`.**

As a reminder, this journal empowers you to:

1. Remember better.
2. Create your own reference book.
3. Make connections.
4. Keep a record of your work.
5. Use your notes to collaborate with others.

**This journal is set up the same way as Karel Jr 1 and 2.**

The journal is divided into sections that match those in the on-line course. Each section has:

1. One or two **review pages** with questions or activities to help you remember what you have learned. There will always be an open box for your own notes.
2. A **bulletin board** where you can post real life examples: paste in pictures, sticky notes, and scribbles. There are ideas and suggestions at the top of each bulletin board.
3. A **planning page** for your end-of-section project.

The back of the journal contains a **glossary with vocabulary from Karel Jr 1 -3**, a **record page** for your files, and a **design template** that can be copied to work on games.

As always, remember to slow down, journal, talk with people, and sketch ideas. We hope you will develop a deeper understanding of what coding is all about, and discover the thrill of having a computer or machine carry out a program that you have written.

Happy coding!



## SECTION 11 NOTES

*Use this space to write your own notes, questions, and problems.*

## QUESTIONS

You are starting to use simple blocks of programming to build complex routines. Explain how this process works in either 11.3/11.4 (using `def star`), or 11.5/11.6 (using `def waterbox`).

11.7 uses three defined commands to create repeated actions. Notice how comments are used as headings and explanations for each defined command, and for the main program. Practice writing a comment and a defined command. Remember to start the comment with the `#` symbol to show that it is just a text string and not part of the program itself.

## SECTION 11 BULLETIN BOARD

<p><i>This is a page to post ideas, pictures, sticky notes, drawings</i></p>	<p><i>Packaging sets of items in containers is a repeated defined function.</i></p> <p><i>Example: premade lunch trays: cheese spread, spreader, salami, crackers, apple sauce, juice drink, straw, napkin are packed into a tray, which is then sealed and moved along a conveyor. How could you program a machine to do this task?</i></p>	

## SECTION 11 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Use at least one `def` keyword to create a defined command, then use it in a program.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:	Maze Sketch (12 rows x 15 columns)	
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		



**SECTION 12: USING DEFINED COMMANDS; ADVANCED MAZE SKILLS**

In this section, you learn that a new command should always be tested on a simple task first, and then it can be safely used as part of a larger program. You also learn advanced maze skills: how to follow a line that is on Karel's left, or one that is on Karel's right.

In 12.1 to 12.3, you practiced a simple task that was repeated in a larger program. What was the simple task, and how was it used?

The defined command `move` is a useful one for irregular mazes. For your own reference, write out the two versions of `move` that Karel uses to follow an irregular path; one for following the wall on his left, and the other for following it on his right. Include indentations so that it is obvious which actions are within the while loop and which are not.



FOLLOW WALL TO THE LEFT	FOLLOW WALL TO THE RIGHT



## SECTION 12 NOTES

*Use this space to write your own notes, questions, and problems.*

## QUESTIONS

George has developed a design that will be used to print Halloween themed fabric.

Here is the pattern, using pumpkin and eye.



Write a defined command design that would make this pattern.

Draw a pattern that could be based on this design.

Write a program that would call a defined command design to make this pattern.

## SECTION 12 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>In Section 12, defined commands could be tasks based on regular patterns that are repeated (like placing the popcorn), or conditional (like following the wall)</i>	<i>When you look for examples, think of both regular patterns and conditional situations.</i>

## SECTION 12 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Use at least one** defined **command** that is used within a larger context. For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:																																																																																																																																																																																				
Program:		Maze Sketch (12 rows x 15 columns)																																																																																																																																																																																				
		<table border="1"> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>																																																																																																																																																																																				
Storyline:																																																																																																																																																																																						
Karel's goals:																																																																																																																																																																																						
Number of steps:	Keywords:																																																																																																																																																																																					
	Required (must use):																																																																																																																																																																																					
Number of operations:	Forbidden (can't use):																																																																																																																																																																																					
Any special challenges:																																																																																																																																																																																						

## SECTION 13: COMPARING PROGRAMS; SOLVING COMPLEX PROBLEMS

In this section, you learn that the shortest program may not always be the best. A slightly longer program that is much faster, is better than a slightly shorter program that takes a lot of time. You know to break a complex problem into smaller tasks which are solved first.

You are at the stage where it is important not just to write code, but also to evaluate it. Here are some criteria.

**Reliability:** does the code work correctly every time? Try all the mazes. Does it work in each one?

**Speed:** does the program work quickly?

**Ease of use:** is the code easy to understand and repair?

**Limitations:** is the code limited to certain conditions? For example, must the path be straight in order for the code to work?

Compare the two programs in 13.1 and 13.2. Write down the number of lines, the number of operations, and the time it takes to run. Add other observations. Decide which one you prefer and explain why using your code checking criteria (reliability, speed, ease of use, limitations).

	13.1	13.2
Number of lines		
Number of operations		
Time to run program		
Observation:		
Observation:		
<b>I prefer _____ because</b>		




## SECTION 13 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Listen to a favorite song or piece of music.</i>	<i>Do you hear structures that could be worked out first, before they are used in the whole piece of music?</i>

## SECTION 13 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Build a program that tests rooms or columns, similar to those in Section 13.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:		Maze Sketch (12 rows x 15 columns)
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		



## SECTION 14: VARIABLES AND FUNCTIONS: INC(), DEC(), AND PRINT

In this section, you learn how to create new variables and initialize them with numbers. You can use the function `inc()` to increase the value of a variable by one, the function `dec()` to decrease the value of a variable by one, and the `print` command to display results. The `print` command can be used to display the values of variables while the program is running.

-1	final	counting	initial
quotation	memory	1	increase
name	decrease	value	strings

Using the word list, fill the blanks in the following definitions.

**Variable:** in terms of programming, variable is the \_\_\_\_\_ and \_\_\_\_\_ of something that will be recorded in \_\_\_\_\_. The \_\_\_\_\_ **variable** is used in Section 14. The \_\_\_\_\_ value of this variable is set: for example:  $n = 0$ .

`inc(n)` tells the program to \_\_\_\_\_ the value of  $n$ . The default increment is \_\_\_\_\_.

`dec(n)` tells the program to \_\_\_\_\_ the value of  $n$ . The default is \_\_\_\_\_.

`print(n)` tells the program to print the \_\_\_\_\_ value of  $n$  after the program has ended. Text \_\_\_\_\_ can be printed out on their own or as part of a command. The text is always enclosed in \_\_\_\_\_ marks.

What were the `inc(n)` or `dec(n)` functions used for in the Section 14 levels?

LEVEL	USE OF INC(N) OR DEC(N)
14.1 to 14.3	
14.4, 14.5	
14.6, 14.7	

## SECTION 14 NOTES

*Use this space to write your own notes, questions, and problems.*

## QUESTIONS

Write the following statements as print commands. The first one is done for you.

Karel has used n coins.

```
print "Karel has used" (n) "coins."
```

n fence sections are damaged.

The total number of pages is n.

All businesses must keep track of items they buy and sell in their inventories. Programs use counting variables to keep track of the increasing and decreasing amounts of each item. It is helpful to write alerts into the program, so that the purchaser knows when to order replacement stock. Write two lines of code that will print out "Order light bulbs, item #10765" if the number of light bulbs is 15.

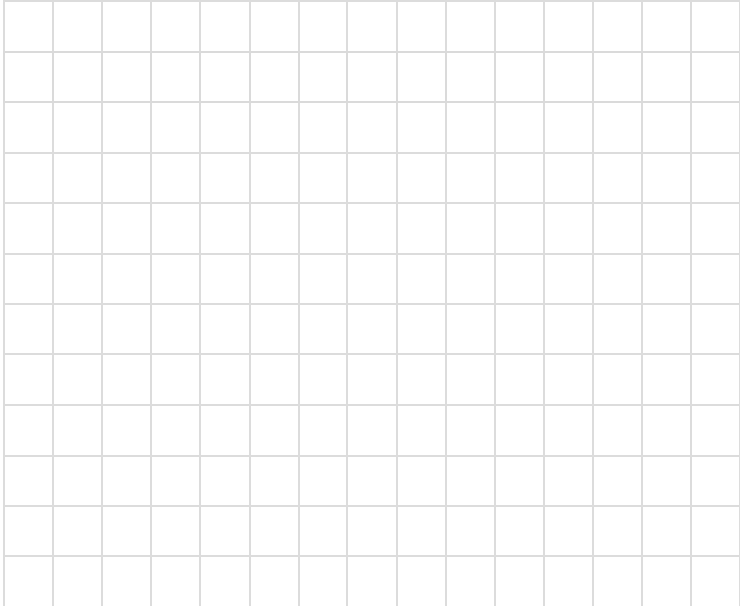
You are completely out of coral bracelets, item #35-672. Write a print message for your website store program which tells the customer that the item is not available. Write it in cheerful language that makes them want to continue browsing your store.

## SECTION 14 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Counting triggers all kinds of events.</i>	<i>Think of how counting and printing messages could be applied.</i>

## SECTION 14 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Use** `inc()` **or** `dec()`, **and use** `print(n)`. For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:	Maze Sketch (12 rows x 15 columns)	
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		

## SECTION 15: VARIABLES AND FUNCTIONS: (INC), DEC(),RETURN, LOCAL/GLOBAL

You learn how to define new functions and return values using the keyword `return`, and use functions `inc()` and `dec()` to increase / decrease the value of a variable by more than one. You know that the value returned from a function can be stored in a variable, and if the returned value is not used, it will be automatically printed. Any code typed after the `return` command is dead. Variables defined **inside** commands and functions are **local**, and local variables cannot be used outside of the command or function where they were defined. Variables created in the main program are **global**, and global variables should not be used inside commands and functions.

Here is some sample code. Underline the local variables, and circle the global variables.

```
def column
# Count pearls:
  c = 0
  while pearl
    right
    go
    left
    inc(c)
  go
  return c

# Main program:
result = column
print "There are", result, "pearls!"
```

Why can't we print the variable `c`?

What is another way to write this program so that we can print `c`?  
(This method is NOT recommended)

## SECTION 15 NOTES

*Use this space to write your own notes, questions, and problems.*


## QUESTIONS

In this Section, you are able to increase or decrease a variable by more than one: in effect, multiplying or dividing for each operation on the variable. Give examples of how you would code the following:


Total sales of bicycles at \$285.00 each	
Students are given pencils from a box of 500. Each student gets 3 pencils.	
Find out how many tomato plants are in each row. Then report on the number of tomato plants in all the rows.	

## SECTION 15 BULLETIN BOARD

<i>This is a page to post ideas, pictures, sticky notes, drawings</i>	<i>Think of other situations that require multiplying and dividing.</i>	<i>If you were gathering information on these situations, what might you want to be alerted to?</i>

## SECTION 15 PROJECT

Use this page to develop ideas for your game for Karel. The person playing your game will need to write code to solve the game. **Use one local and one global variable. Try increasing or decreasing the amount by more than one.** For your notes, you can draw the solution paths through the maze, and write the correct code to the left. **Just make quick notes here: you can make more detailed notes and description in Designer Mode using Edit Game.**

Game Name:		Date:
Program:		Maze Sketch (12 rows x 15 columns)
		
Storyline:		
Karel's goals:		
Number of steps:	Keywords:	
	Required (must use):	
Number of operations:	Forbidden (can't use):	
Any special challenges:		



## REVIEW YOUR PROGRESS

This is the final Section of Karel Jr 3. Reflect on what you have learned so far.

Rate yourself C, B, or A:

- C if you could use this skill any time and could coach someone else;
- B if you have a good understanding but need more practice, and
- A if you feel that you are unsure of yourself and need teaching or coaching.

SKILL OR CONCEPT	C	B	A
Use the <code>def</code> keyword to create a defined command			
Use defined commands more than once in a program			
Build a complex task out of several defined commands			
Follow walls to the left or to the right			
Compare and evaluate different programs that carry out the same task			
Use <code>inc()</code> and <code>dec()</code> in default mode, and counting by any amount			
Use the <code>print</code> command and text strings			
Use functions, local and global variables, the return command.			

**Now, set some learning goals based on your self-evaluation. Don't worry if you aren't an expert yet!**

<b>RETAKE CERTAIN LEVELS</b> <b>FIND A COACH</b> <b>REVIEW AND DISCUSS NOTES</b> <b>PRACTICE</b>	<b>PRACTICE</b> <b>REVIEW AND DISCUSS NOTES</b> <b>CREATE</b>	<b>READY FOR THE NEXT COURSE</b> <b>CREATE COACH</b>

## LIST OF BASIC COMMANDS AND KEYWORDS FROM KAREL JR 1, 2, AND 3

Command words: `go`, `left`, `right`, `get`, `put`

Directional commands (`go`, `left`, `right`) are always from the robot's point of view.

`go` advances the robot one step.

`left` turns the robot to its left.

`right` turns the robot to its right.

Retrieving and placing objects (`get`, `put`)

`get` picks up an object

`put` places an object

Loops

`repeat x`, where `x` = the number of times the command is to be repeated.

`while x`, where `x` = a defined condition

Conditions

`if x`, where `x` = a defined condition (this may be a single sensor word or a more complex set of conditions using sensor words and operators)

`else` may follow an `if` condition to provide the alternative course of action

Keywords that are Logical Operators

`not` the condition is that the sensor is `not` present

`and` the condition needs all of the sensors joined by `and` to be present

`or` the condition needs one of the sensors joined by `or` to be present

Important Sensor Words (Karel senses objects and containers when he is in the same square.)

`empty` Karel's pocket is empty

`north` Karel is facing `north`, or the top of the maze grid.

`wall` any obstacle is a type of `wall`. Karel senses it in the square in front of him.

`home` the `home` square. Karel sense it when he is in that square.

Defined commands

`def` `def` begins a defined command, which is a set of commands that will be called in the main program.

## Working with functions and variables

`inc(n)` tells the program to increase the value of `n`. The default increment is 1. To increase by a different amount, write the function as `inc(n, x)`, where `x` is either a value or a variable that will increase `n`.

Example:

`inc(n, 2)` increases the variable `n` by 2 each time

`inc(total, r)` increases the variable `total` by the variable `r` each time.

`dec(n)` tells the program to decrease the value of `n`. The default is -1. To decrease by a different amount, use the same rules as for increasing.

`print(n)` tells the program to print the final value of `n` after the program has ended. Text strings can be printed out on their own or as part of a command. The text is always enclosed in quotation marks.

Example:

`Print "Placed one bottle."` will print "Placed one bottle."

`Print (n) "bottles remain."` will print "36 bottles remain." (if `n=36`)


`return n` ends the function, returning a final value for the variable.


## LIST OF KEY VOCABULARY FROM KAREL JR 1, 2, AND 3 (IN ORDER OF APPEARANCE)

**Command words:** `go`, `left`, `right`, `get`, `put`. These words tell Karel what to do.

**Home** is the destination square, marked by red diagonal stripes which change to green when Karel approaches the square. The word `home` is also used in conjunction with commands.

**Max** may refer to maximum number of steps, operations, or programming lines.

**Steps** are the number of squares that Karel moves. The shoe icon  counts the number of steps.

**Operations** are anything that Karel does: move, turn, pick up or put down objects. The computer icon  counts the number of operations.

**Objects** are items placed in the maze. (The word “object” can have other connotations in programming that are not used here).

**repeat** is written on its own line as `repeat x`, where `x` = the number of times the command is to be repeated.

**Body:** the body contains the commands to be repeated. The commands are written on the lines following the `repeat` command, indented two spaces.

**Loop:** A set of commands repeated a given number of times.

**Nested loop:** A loop that is within another loop.

This is a good time to introduce some of the terms used in programming. Refer to the online textbook under Section 5 Programming for details.

**Algorithm:** a series of logical steps that leads to the solution of a task. Students may be familiar with algorithms used in operations such as subtraction and long division.

**Logical error:** a mistake in an algorithm. Planning helps reduce the number of errors.

**Computer Program:** An algorithm written using a programming language.

**Syntax:** the way a command line is written.

**Syntax error:** a mistake in spelling, operators, indentations, spaces

**Sensor words:** items from the Karel library, which can include collectible items (such as `orchid`), containers (such as `basket`), and obstacles (such as `wall`, `plant`). A word that is both in the library and correctly spelled will be blue-colored. Collectible and container items are sensed in the square that Karel occupies. Obstacles are sensed in the square in front of Karel.

**if** is written on its own line as `if x`, where `x` = a defined condition. In these lessons, predefined objects from the library are used as sensor words for the condition.

The body contains the commands to be followed if the `if` condition is met. The commands are written on the lines following the `if` condition, indented two spaces.

**Condition (Section 8 in the textbook):** tells the program what to look for and how to act. Conditions make decisions while the program is running and handle unexpected situations. The program may need to collect all the coins it finds, but may not know where the coins will be located. The `if` condition says: "Is there a coin? If there is a coin, get it." Conditions work like a switch. Note: because `if` conditions test each instance separately, they are NOT loops, even though they are written with a similar format.

**Satisfy:** in programming, satisfy means to meet the condition - the condition exists.

**Aisle:** a row or column with objects on either side

**Sensor:** the presence of something, such as a coin, used to create a condition.

`north`: "`if not north`" can be used to detect if Karel is facing north (the top of the maze), and can be used to reorient Karel to any direction, once he is facing north.

**Key words** `or`, `and`, `not`:

`or`, `and`, `not` are logical operators for the condition. In order to execute the command, `or` means that one (or a set of conditions within parentheses) of two or more conditions must be met, `and` means both or all of the conditions must be met, `not` means that condition must not be met.

`empty`: tells whether or not the robot has an object in its pocket. This creates a condition, either `if empty`, or `if not empty`

`while`: A `while` loop is a repeated set of commands that will continue as long as the condition being sensed is present. The number of repetitions is not known in advance. The `while` loop continues until the condition is no longer sensed. `while` loops use the same sensors as `if` conditions. A `while` loop is different because it continues until the condition is no longer sensed, whereas the `if` condition senses each square as a separate test.

**Infinite loop:** If a loop never senses when to end (the stopping condition), it can continue infinitely. Fortunately, most programs will time out if this happens. In Karel, programs can always be stopped manually if this happens.

**Defined commands:** `def` begins a defined command, which is a set of commands that will be called in the main program. Defined commands can be used as many times as needed in the

main program. This can reduce the number of lines needed, and also makes editing easier. If there is an error in the defined command, it can be fixed in one place.

**Text string:** words included in the program that are descriptive and not part of a command. Text strings are enclosed in quotation marks and are separated from command words by a comma.

**Comment lines:** lines of text strings, always starting with the # sign that describe what is happening in the program. Quotation marks are not needed in this case.

**Variable:** in terms of programming, variable is the **name** and **value** of something that will be recorded in **memory**. The **counting variable** is used to increase or decrease a value

**Function:** a defined command or set of commands based on a variable that returns a value. Functions `inc()` and `dec()` are used to increase or decrease a variable by a specified value.

**Local variable:** a variable created within a command or function. A local variable cannot be used outside of that particular command or function.

**Global variable:** a variable created in the main program. A global variable cannot be used inside of a command or function.

**Return:** the `return` command ends the function, returning a final value for the variable.

## FILE LOG: GAMES I HAVE CREATED

FILE NAME AND LOCATION	DATE	DESCRIPTION	NOTES
1.			
2.			
3.			
4.			
5.			
6.			
7.			
8.			
9.			
10.			

## DESIGN TEMPLATE

Game Title: \_\_\_\_\_ Date: \_\_\_\_\_ Author: \_\_\_\_\_


**Story Ideas:**

**Maze elements:**

**Programming ideas:**



## NOTES