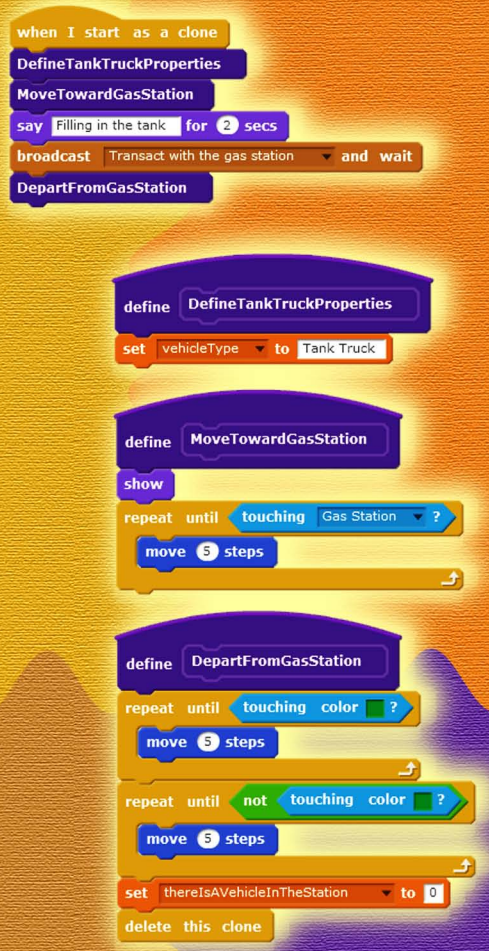
G. An. Papadopoulos
D. G. Fotiadis
An. N. Ladias

Student Book

Special programming topics in Scratch



Writing hierarchical modular object-based event-driven code in a visual block-based programming environment

ΕΛΛΗΝΟΓΕΡΜΑΝΙΚΗ ΑΓΩΓΗ
ELLINOGERMANIKI AGOGI
inspiring SCIENCE education

Special programming topics in Scratch

Writing
hierarchical, modular, object-based, event-driven
code
in a visual block-based programming environment

Student Book

Writers

Georgios An. Papadopoulos
gpap@ea.gr

Dimitrios G. Fotiadis
dimitris.fotiadis@gmail.com

Anastasios N. Ladias
ladiastas@gmail.com

We would like to extend our warmest thanks to Dr. Sofoklis Sotiriou, head of Ellinogermaniki Agogi's R&D department, for his scientific aid in our project, and to his department for the technical and administrative support it provided us during the authoring of this book.
We would also like to extend our warmest thanks to Mrs Mariangella Tsangari, English teacher of Ellinogermaniki Agogi, for her generous help in translating the book's preface.

The creation and reproduction of the book "*Special programming topics in Scratch / Writing hierarchical, modular, object-based, event-driven code, in a visual block-based programmingenvironment / Student Book*" is co-financed by the European Commission within the framework of the project: "InspiringScience: Large Scale Experimentation Scenarios to Mainstream eLearning in Science,Mathematics and Technology in Primary and Secondary Schools", with contract number 325123.

ISBN: .....

Contents

Introduction

The educational field has witnessed a global shift of focus on programming and this becomes apparent via events such as the Hour of Code, the Europe Code Week etc. This shift seems to be imposed by the ever growing demand for computer literate people, skilled at programming in a rapidly developing software industry.

With regard to our country, Greece, considering the economic crisis, the software industry could serve as a way out from recession; as this sort of industry needs no costly investments for industrial premises and equipment but highly trained and specialized staff, who will be able to be innovative, inventive and resourceful in terms of new pioneering ideas which will be transcribed as a code and hence, the initial idea will be transformed into a final product. The aforementioned characteristics prove to be consistent with the mentality of modern Greeks, the descendants of Ulysses, as it becomes obvious by their global and European rankings for the past two years in the Hour of Code and the Europe Code Week.

It may be the case that this international coincidence serves as an opportunity for Greece to actively participate in a globalized cutting-edge technological trend.

As a result of this trend aiming at the promotion of the code, numerous exceptional educational packs (software, books, MOOCs) have been created in order to introduce the audience to programming in an easy and visual way. However, the majority of this educational resources still focus on teaching students how to handle rather simple algorithms by using selection and iterationcommands. On the other hand, a plethora of scientific educational resources are available that serve the purpose of professional programming. Hence, what the present book is trying to achieve is to bridge the gap existing in between simple programming and university teaching resources.

It is our firm belief that the reader knows not only how to solve simple algorithms but also how to write the respective programs in a visual programming environment (such as Scratch, App Inventor etc). Via the use of genuine and meaningful scenarios this book focuses on creating a mental scaffolding that will help the student tackle more complex problems in an easy way, by coding in an object-based, event-driven, hierarchical modular pro-

gramming environment. For this purpose, Scratch was employed as the programming environment which offers simultaneously to students and teachers, a pleasant, free and solid visual programming tool, which providesnot only all modern programming structures in accordance with hierarchical modular programming principles but also the power and flexibility of event driven programming and programming messages that may be exchanged by the objects of our application.

The majority of algorithms existent in the present book have been selected from the syllabus examined in the Panhellenic Exams for the 'Developing applications in a programming environment' school subject of the final school grade. Moreover, the aforementioned syllabus is adjusted to the particular features of the Scratch environment. What follows is the step-by-step verbaldescription of the complete functionality of the system and the resulting description of the actions to be performed by each object of the program. This will eventually be transformed into Scratch code.

The problems were of a complex nature and this resulted inevitably in complex code. The need to not only handle but also teach to our students the complexity of composite problems led to the invention of 'CodOrama' (from the Greek words Κώδιξ/Code and Οραμα/Orama). This need could be defined through the overall perception of the code's totality on the one hand and, at the same time, the focusing on the significant details of the different modules that comprise the objects in different states, on the other hand.

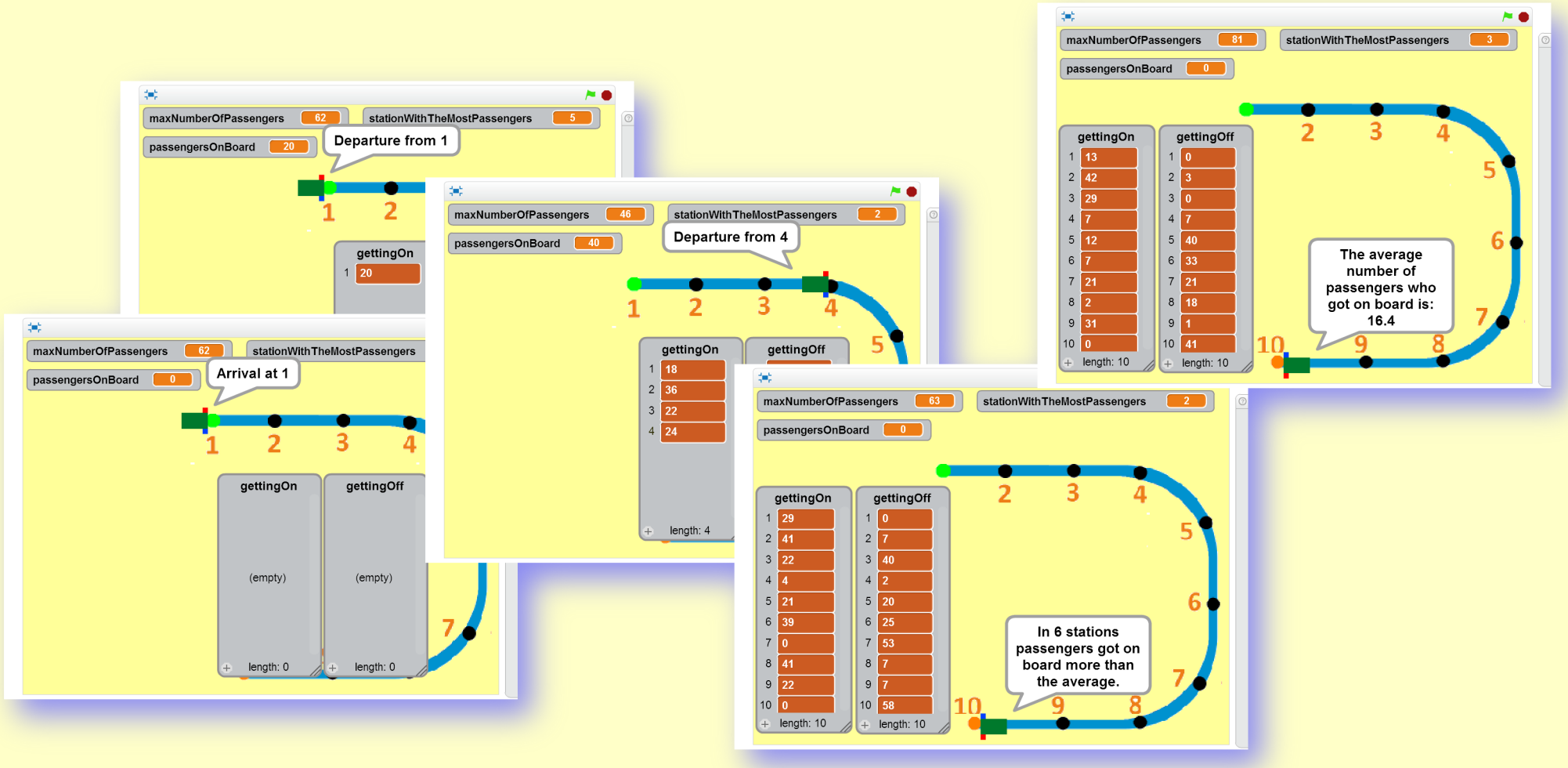
'CodOrama'is a two-dimensional table. On one dimension there are the objects of the system and on the other the states where the objects can be found during the execution of the system. On the thus defined surface, cells are created that contain the visualized code determining the behavior of the correspondingobjects in the corresponding states. Communication is performed via variables and messages providing the ability for interaction between code modules of one object with code modules of another object. The supervision offered by 'CodOrama' enables the student to distinguish the code modules that are performed sequentially from those performed in a parallel way.

Also, in every code of the book internationally established legible code conventions are adopted and so, the need for additional explanatory comments is significantly limited.

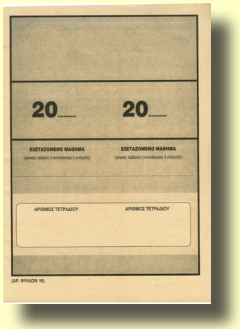
In 1990, in the introduction of the book ‘Pascal, a methodical approach’ (An. Ladias, Kledarithmos) we wrote: “The aim of this book is to ‘infect’ students with the virus of the programmer, who enjoys a nice program when unfolded, feels satisfaction by recognizing the well-chosen variable names, enjoys creating smart and flexible sub-programs, is thrilled by the pleasant absence of anything redundant ...” Today, 25 years later (a period of time that seems an eternity for informatics) we would not change anything to the aforementioned introduction.

Coding is not easy endeavor, yet it is a creative one that entails not only labor but genuine pleasure for the creator.

1. Train passenger management



Hellenic National Examinations Question



Secondary School Leaving Examination
May 27th 2009
Tested course: Developing applications in a programming environment

Question 3

In a train route there are 20 stations (including the start and the terminal stations). The trains stops at every station where passengers board and disembark. The first passengers board at the start station and all passengers on board disembark at the terminal station.
Write an algorithm that will manage passenger flow. Specifically the algorithm should:

1. Ask the user to enter the number of passengers getting aboard in each station except for the terminal station, and then insert that number into the array GETON[19].

2. Insert into the array GETOFF[19] the number of passengers disembarking in each station, except for the terminal station, as follows: for the start station insert the value zero (0); for all intermediate stations ask the user to enter the number of passengers getting off the train.

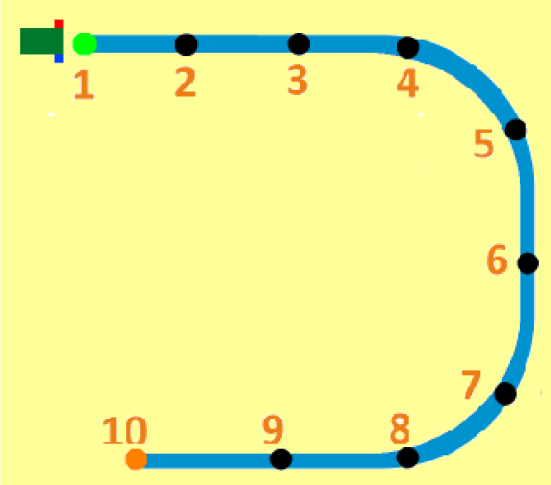
3. Create an array PASS[19] where it should store the number of passengers aboard after the train departs from each station.

4. Find and show the station which the train is departing from with the maximum number of passengers. (Assume that the train departs from each station with a different number of passengers.)

Hellenic National Examinations Question - Tested course: Developing Applications in a Programming Environment - Scratch adaptation

The itinerary of a Train includes a starting point, 8 intermediate stations and the terminal station.

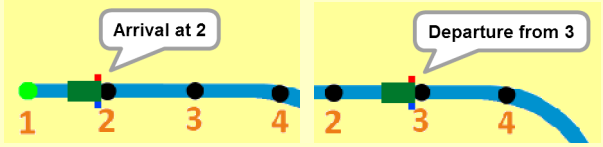
The locations of the intermediate stations and the terminal are not known in advance to the Train, but it can instead trace the blue path of itsitinerary by using its sensors (located in the front of it), and also locate the stations by their color (black color for the intermediate stations and orange color for the terminal station).



The first passengers are embarking at the starting station and then the train starts moving along its path.
When the Train arrives at an intermediate station it stops to board and disembark passengers.
When theTrain arrives at the terminal station, all passengers disembark and the train's itinerary is completed.
The maximum number of passengers allowed to get on the train at each station is 50.

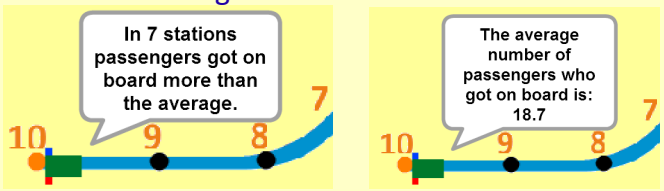
The program must implement the following:

1. The arrival and departure of the Train at each station must be announced along with the station's serial number (or ID).



2. When the train is at a station the program must remember how many passengers got on and how many got off.

3. When the train arrives at the terminal the program must announcethe number of stations where the passengers who boarded the train were more than the average number boarding over all stations and also what this average number is.



Program Control Flow and Sprites' Functional Analysis

1. When the user clicks the green flag, the Train performs the following tasks at the start station:

- **It initializes the variables and lists it is using,**
- **sets up its appearance,and**
- **finally calls the Arrival procedure (see step 4).**

2. Next the FindMaxNumberOfPassengersAndStationWithTheMostPassengers procedure is being called that determines the maximum number of passengers over all stations that have thus far been visited by the Train, as well as the ID of the station in which this maximum number occurred.

3. Now we enter a loop until the Train touches the orange color of the terminal station. In that loop we repeat the following procedures:

- **TrainMotion: the Train uses its blue and red sensors to swerve along its path, and at the same time makes 3 steps in each iteration.**
- **FromTrainArrivalToDeparture: if the Train enters a station (i.e., if it touches its black color), we call the Arrival procedure (see step 4), making sure not to call it more than once while the Train is still in the station (i.e., still touching its black color). To our aid comes the use of the arrivalflag variable which we raise once the Train arrives at the station (when it first touches its black color) and lower as soon as we leave it (i.e., while we don't touch the black color).**
- **At the end of each iteration we call the FindMaxNumberOfPassengersAndStationWithTheMostPassengers procedure.**

4. The Arrival procedure initially increments by one the station id, announces the arrival of the train at the current station and then calls the GettingOffGettingOn procedure. After this procedure has been called, Arrival announces either the departure of the train (if the current station is an intermediate one), or the end of the itinerary (if the current station is the terminal).

5. The GettingOffGettingOn procedure does the following at each station:
It initially specifies the passengers getting off the train: if the station is the starting one, the passengers getting off are 0. If the station is the terminal one, passengers getting off are all the passengers of the train, and finally, if it's an intermediate station, the passengers that get off the train are a random number between 0 and the current passengers.

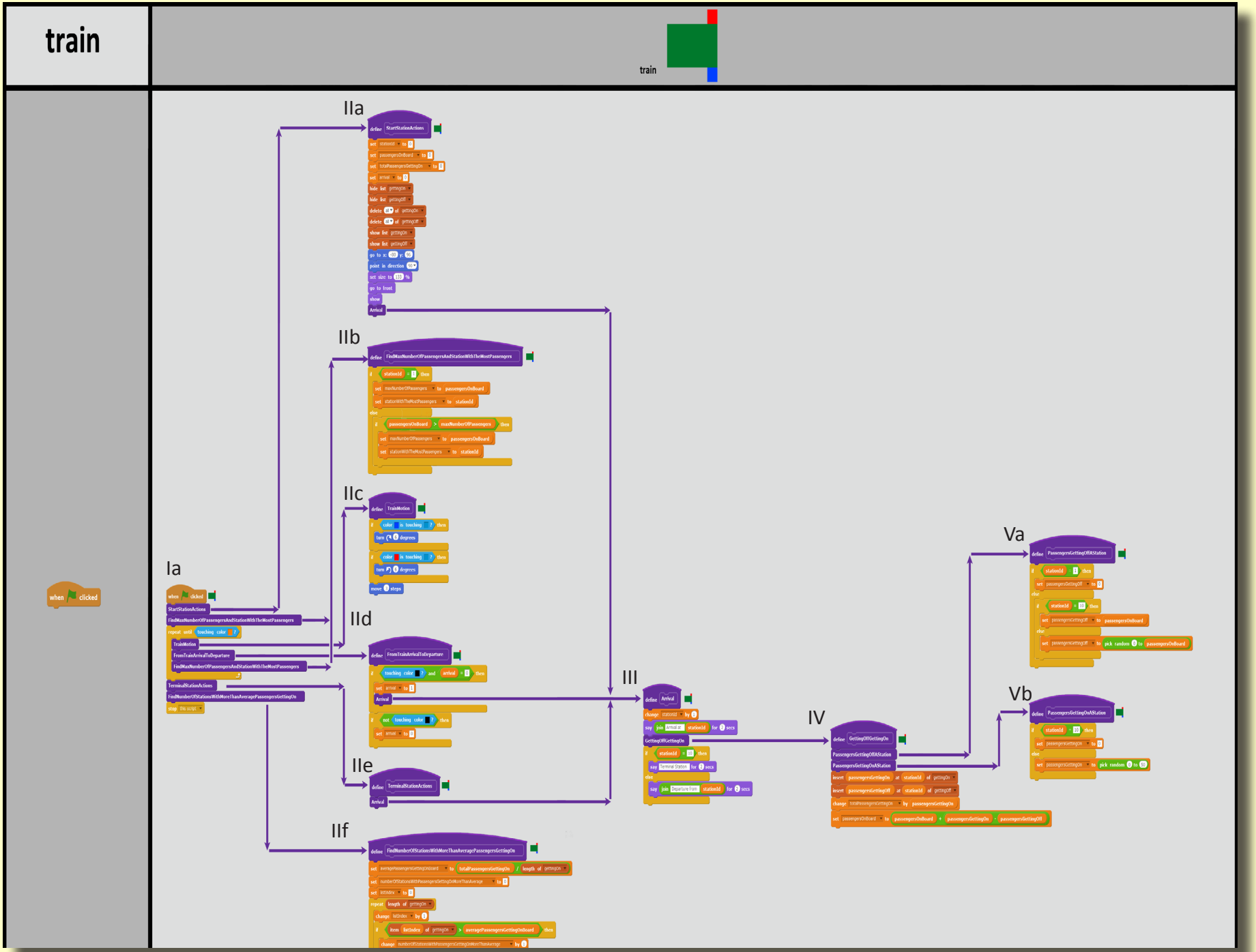
- It then goes ahead and specifies the passengers getting on the train: if we are talking about the terminal station, those getting on are 0, whereas for all other stations the passengers getting on are taken to be a random number between 0 and 50.
- Then we proceed to store the number of passengers getting on the train at the current station into the gettingOn list (at the position of the list indicated by the id of the current station), and accordingly store the number of passengers getting off the train in the gettingOff list.
- Finally, the GettingOffGettingOn procedure updates the total number of passengers across all stations getting on the train, as well as the number of passengers currently on board just before the train departs from the station.

6. Once the Train arrives at the terminal station, we exit the loop and execute the TerminalStationActions, which does nothing more than call the Arrival procedure.

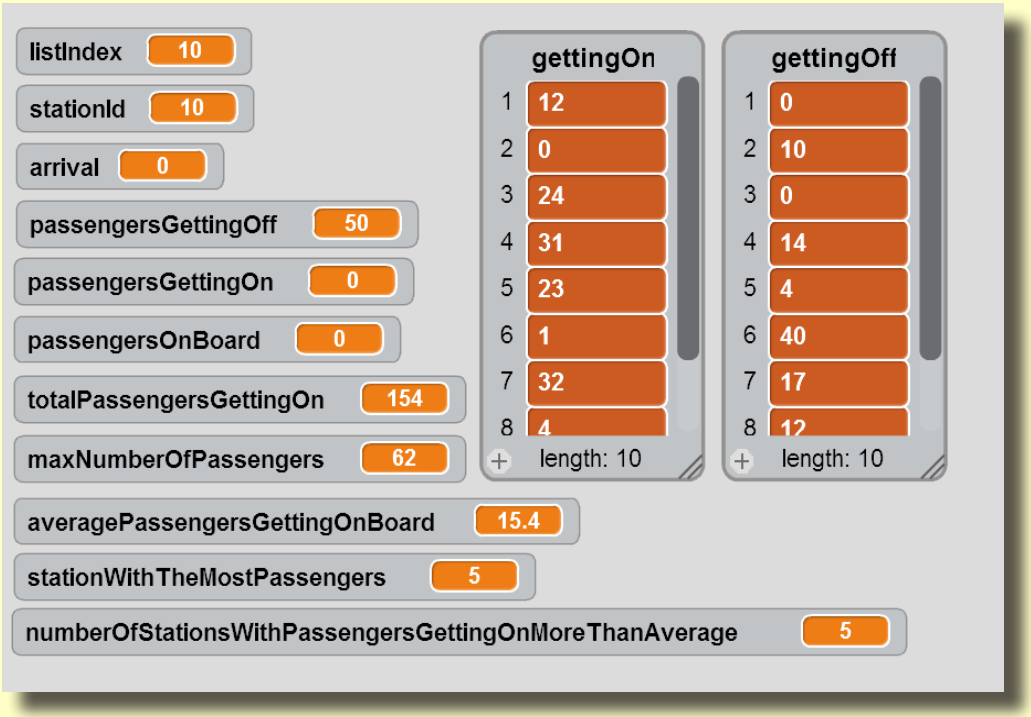
7. As a last action before completing the program, we run the FindNumberOfStationsWithMoreThanAveragePassengersGettingOn procedure.

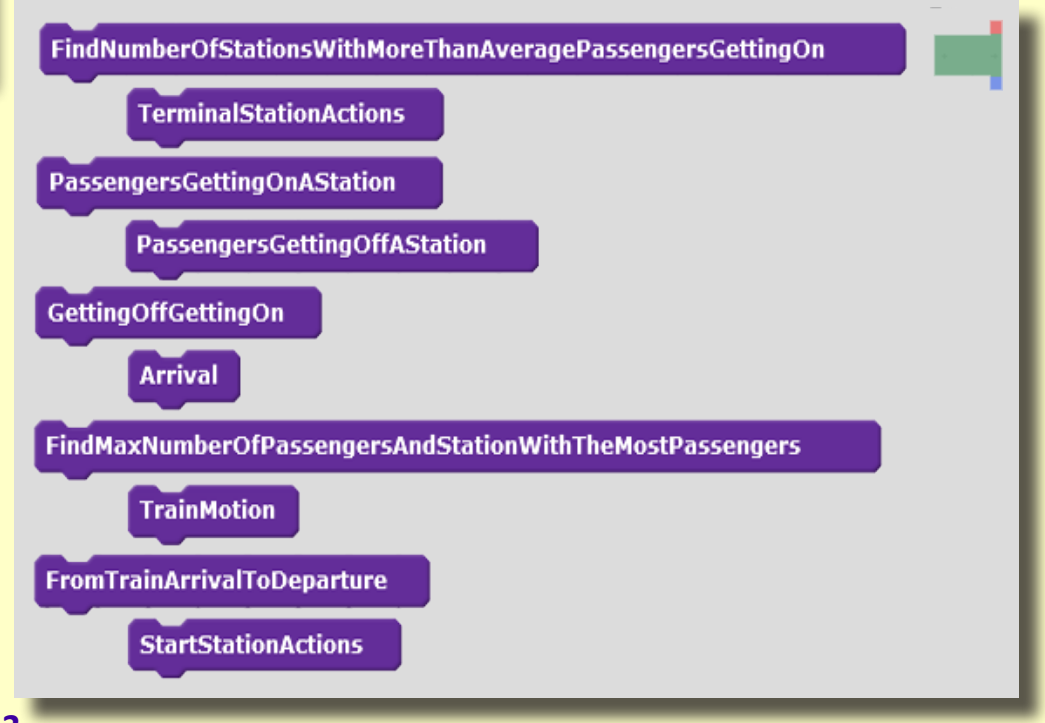
CodOrama

The table on this page, where we can see the various code modules and their interconnections, may be used in order to understand these scenarios.



Variables - procedures





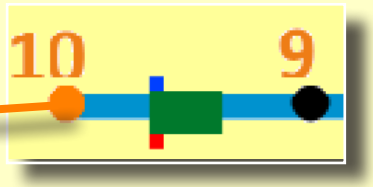
The code in Scratch

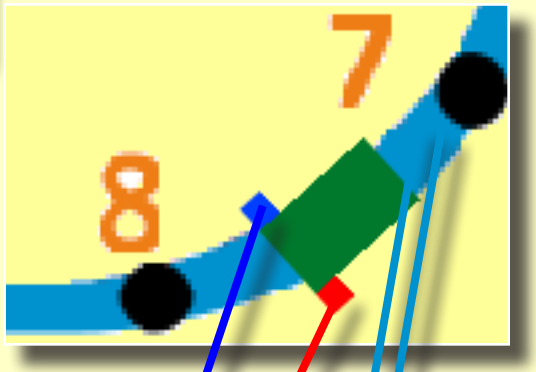
CodOrama: https://prezi.com/qyr0afmrky7l/train/ or http://bit.ly/1jEo61e

Code: https: https://scratch.mit.edu/projects/82678432/ or http://bit.ly/1Lwcvfs

Ia

when clicked
StartStationActions
FindMaxNumberOfPassengersAndStationWithTheMostPassengers
repeat until touching color ?
TrainMotion
FromTrainArrivalToDeparture
FindMaxNumberOfPassengersAndStationWithTheMostPassengers
TerminalStationActions
FindNumberOfStationsWithMoreThanAveragePassengersGetting On





IIb

define FindMaxNumberOfPassengersAndStationWithTheMostPassengers
if stationId = 1 then
set maxNumberOfPassengers to passengersOnBoard
set stationWithTheMostPassengers to stationId
else
if passengersOnBoard > maxNumberOfPassengers then
set maxNumberOfPassengers to passengersOnBoard
set stationWithTheMostPassengers to stationId

IIc

define TrainMotion
if color is touching ? then
turn 6 degrees
if color is touching ? then
turn 6 degrees
move 3 steps

Ia

gettingOff

gettingOn

gettingOn

(empty)

length: 0

gettingOff

(empty)

length: 0

IIa

define StartStationActions

set stationId to 0

set passengersOnBoard to 0

set totalPassengersGettingOn to 0

set arrival to 0

hide list gettingOn

hide list gettingOff

delete all of gettingOn

delete all of gettingOff

show list gettingOn

show list gettingOff

go to x: -80 y: 96

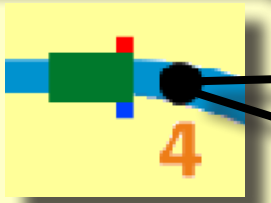
point in direction 90

set size to 133 %

go to front

show

Arrival



IIe

define FromTrainArrivalToDeparture

if touching color ? and arrival = 0 then

set arrival to 1

Arrival

if not touching color ? then

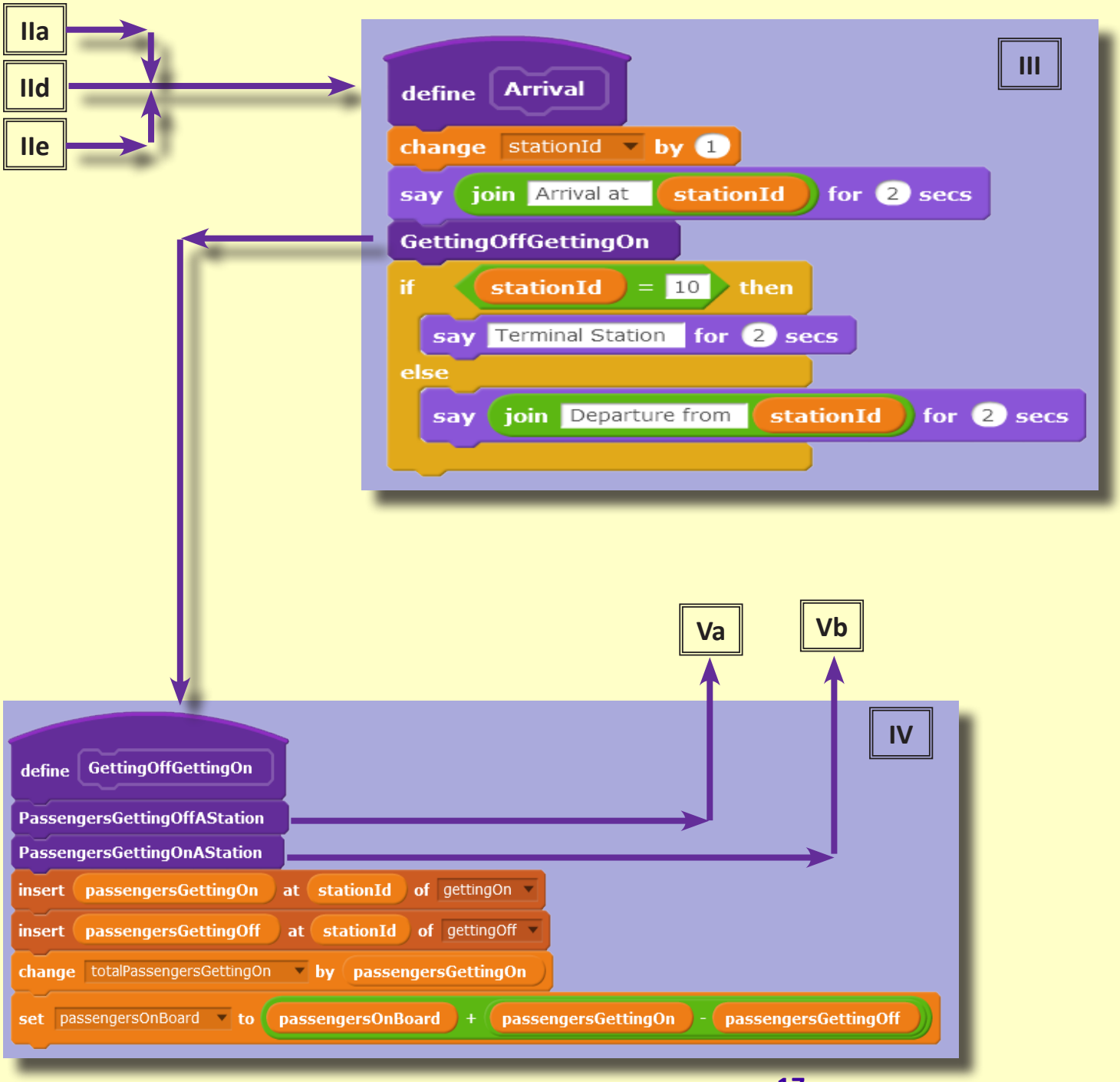
set arrival to 0

IIe

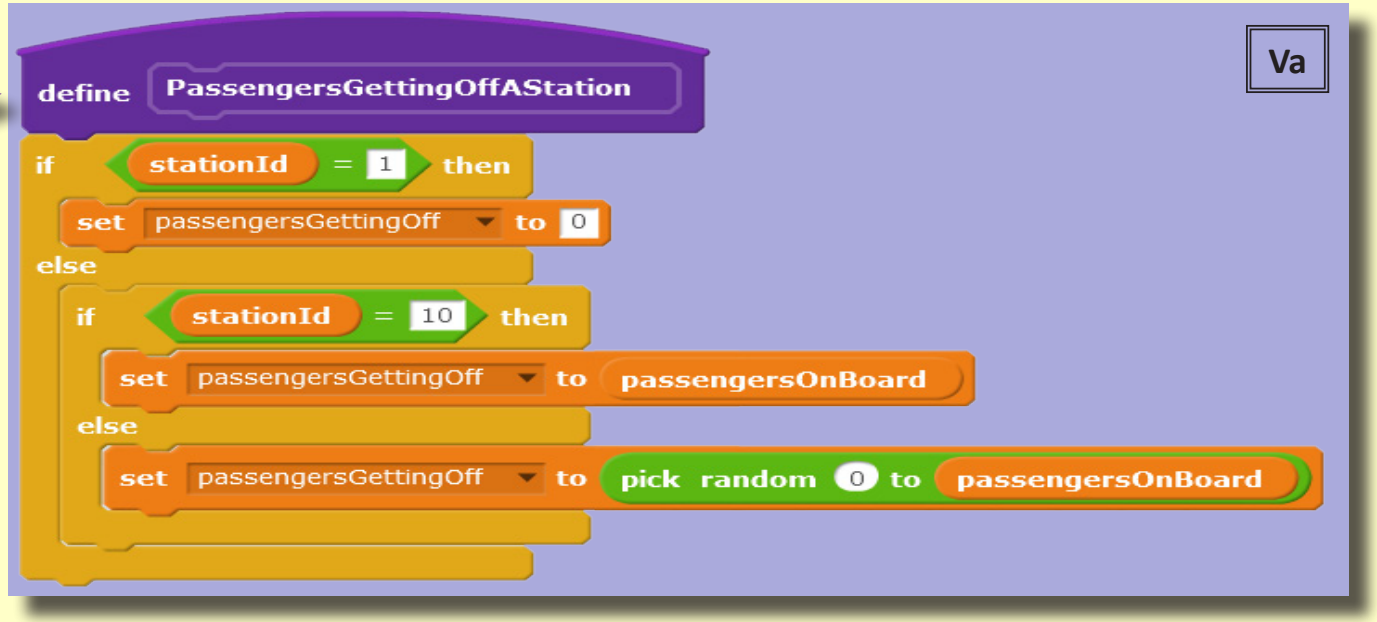
define TerminalStationActions

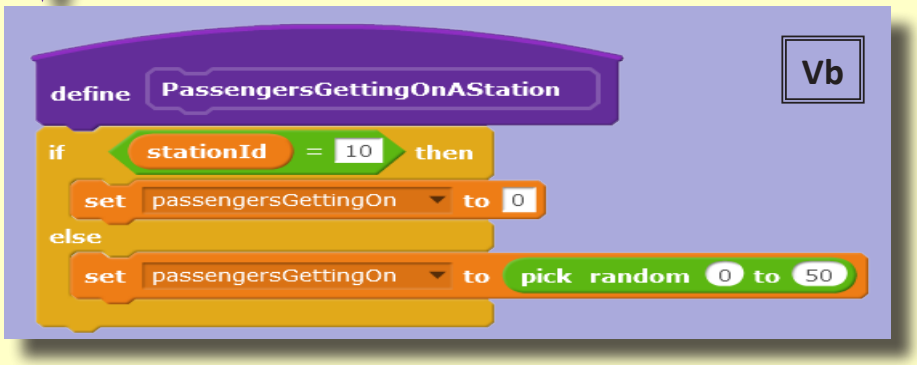
Arrival

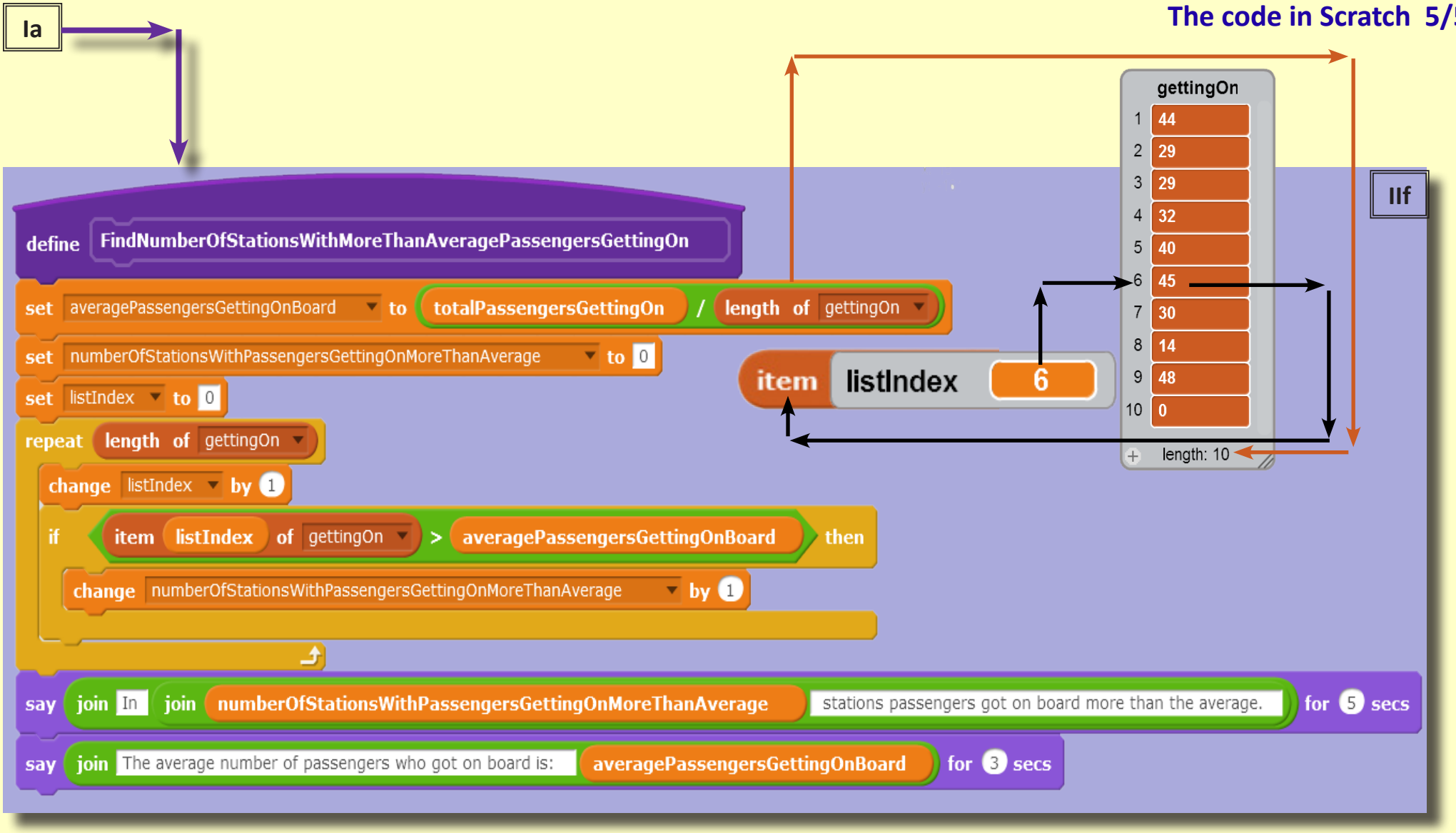
III


IIa
IId
IIe
III
define Arrival
change stationId by 1
say join Arrival at stationId for 2 secs
GettingOffGettingOn
if stationId = 10 then
say Terminal Station for 2 secs
else
say join Departure from stationId for 2 secs
Va
Vb
IV
define GettingOffGettingOn
PassengersGettingOffAStation
PassengersGettingOnAStation
insert passengersGettingOn at stationId of gettingOn
insert passengersGettingOff at stationId of gettingOff
change totalPassengersGettingOn by passengersGettingOn
set passengersOnBoard to passengersOnBoard + passengersGettingOn - passengersGettingOff

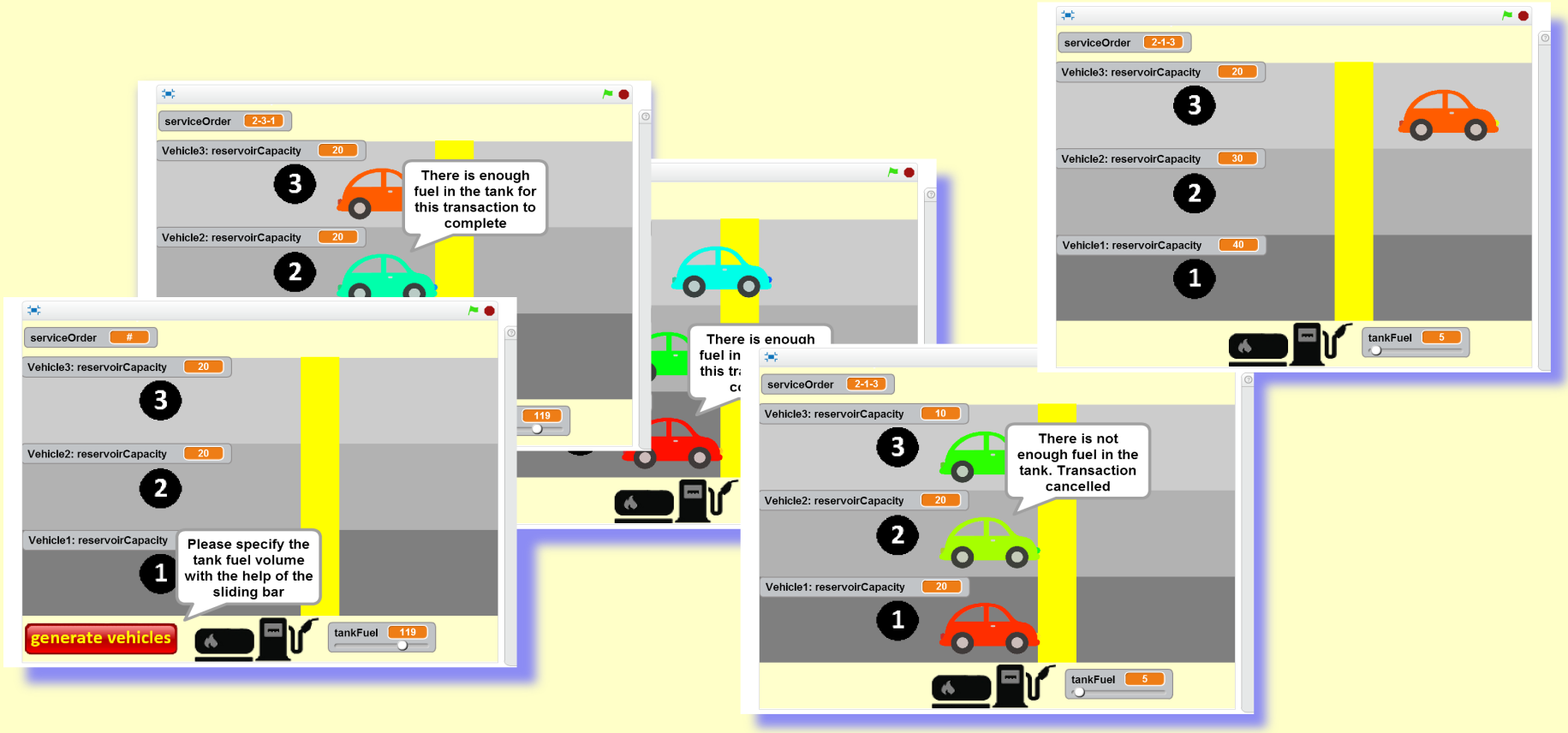
IV


Va
define PassengersGettingOffAStation
if stationId = 1 then
set passengersGettingOff to 0
else
if stationId = 10 then
set passengersGettingOff to passengersOnBoard
else
set passengersGettingOff to pick random 0 to passengersOnBoard


Vb
define PassengersGettingOnAStation
if stationId = 10 then
set passengersGettingOn to 0
else
set passengersGettingOn to pick random 0 to 50


Ia
define FindNumberOfStationsWithMoreThanAveragePassengersGettingOn
set averagePassengersGettingOnBoard to totalPassengersGettingOn / length of gettingOn
set numberOfStationsWithPassengersGettingOnMoreThanAverage to 0
set listIndex to 0
repeat length of gettingOn
change listIndex by 1
if item listIndex of gettingOn > averagePassengersGettingOnBoard then
change numberOfStationsWithPassengersGettingOnMoreThanAverage by 1
say join In join numberOfStationsWithPassengersGettingOnMoreThanAverage stations passengers got on board more than the average. for 5 secs
say join The average number of passengers who got on board is: averagePassengersGettingOnBoard for 3 secs
item listIndex 6
gettingOn
1 44
2 29
3 29
4 32
5 40
6 45
7 30
8 14
9 48
10 0
length: 10
IIf

2. Priority Service Management with Semaphore



Scratch as a multitasking environment: Critical sections and Mutual Exclusion

As we know, many shared resources may be found in a multitasking computer system.
It is very important to determine the way in which the different processes of a program manage these resources, because a change in the data caused by a process may also affect other processes of the same program.

A typical case of processes sharing data is an automated system that supplies fuel to vehicles through a gas station's single pump. The vehicle reaches the fuel pump and initially checks whether the requested amount of fuel exists in the gas station tank.

If the fuel quantity requested by the vehicle is less than or equal to the amount of the gas station tank fuel

then:

- **the vehicle announces that the transaction can be executed**
- **the amount of the tank fuel is decremented by the amount of fuel requested by the vehicle**

else

- **the vehicle's refueling cannot be done.**

But what will happen if two or more vehicles request fuel from the pump at the same time? In this case two or more cars will be served if the tank quantity is adequate. Otherwise the system will not refuelany of the requesting vehicles.

Is the automated system capable of managing an event like this?

Time slot	Quantity of fuel requested byVehicle A: 30 lt	Quantity of fuel requested byVehicle B: 20 lt	Gas station tank fuel quantity
1	---	---	40
2	Is gas station tank fuel quantityadequate? **yes**	---	40
3	---	Is gas station tank fuel quantityadequate? **yes**	40
4	**Message**: The transaction will be carried out	---	40
5	---	**Message**: The transaction will be carried out	40
6	Gas station tank fuel quantitychanges to: **40-30=10**	---	10
7	---	Gas station tank fuel quantity changes to: **10-20=-10**	**-10**
8	---	---	

Therefore we notice that when a vehicle executes the code in its Critical Section, no other vehicle should execute its own Critical Section. In other words, there is a need for Mutual Exclusion of related processes.
In order to solve this problem, we will use semaphores. Semaphores are used whenever we need Mutual Exclusion of processes that execute their code in parallel.

Hellenic National Examinations Question - Tested course: Developing Applications in a Programming Environment - Scratch adaptation

The owner of a gasoline station wishes to automate the refueling of smart Vehicles.

A smart vehicle may:

Know:

The capacity of its reservoir,

The quantity of fuel in the station's tank

Execute the transaction between it and the gas station.

The functional requirements of the program are the following:

a. If the gasoline quantity in the station tank is enough for refueling the Vehicle

then

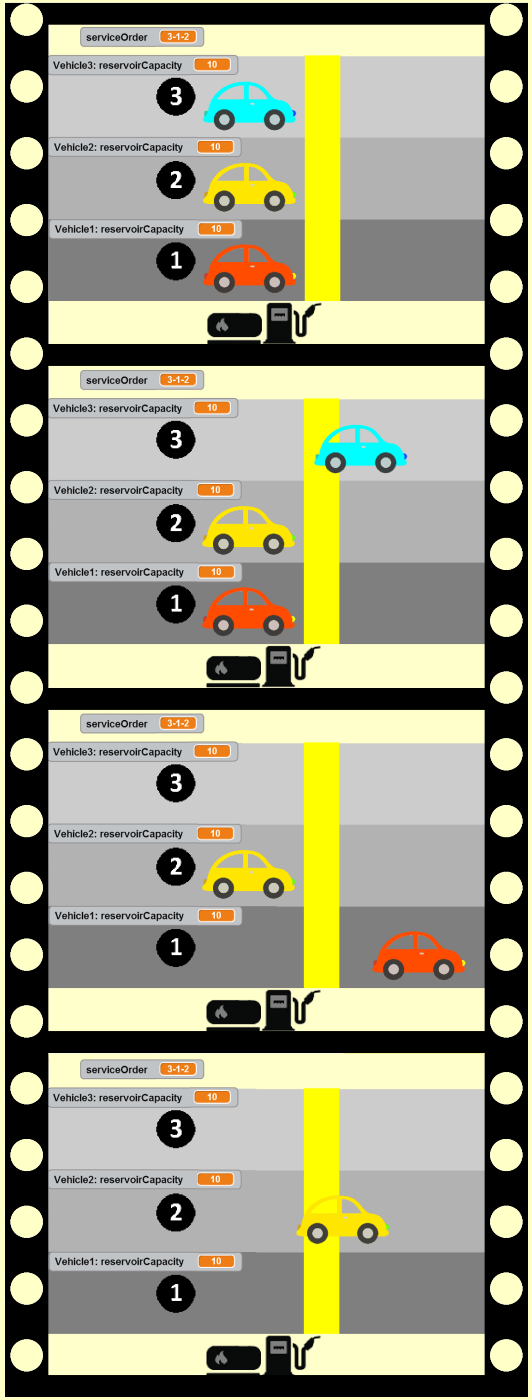
the Vehicle gets refueled (i.e., we decrement the tank gasoline quantity by the capacity of the Vehicle's reservoir),

otherwise

The Vehicle says it cannot be served.

In either case, after the above transaction with the gas station the Vehicle leaves the station.

b. The program must allow the simultaneous arrivals of the three vehicles and their successive service. The gas station has three refueling lanes (one for each vehicle) and a pump connected to its tank.



Program Control Flow

1. When the user clicks the green flag, each of the three vehicles (Vehicle 1, Vehicle 2, Vehicle 3) hides and then positions itself at the start of the corresponding lane and sets up its attributes (a random color for its appearance, and a random reservoirCapacity from the set {10, 20, 30, 40})..

2. When the user clicks the green flag, the Vehicle Creation Button initially notifies the user that they must specify the tank fuel volume (by using the sliding bar under the tankFuel variable), initializes the numberOfVehicles counter to 0 and then broadcasts the message "Count Vehicles", waiting for all interested sprites to handle it.

3. Each of the three vehicles, upon receiving the message "Count Vehicles", increments the numberOfVehicles counter by 1.

4. Subsequently the initial scenario of the Vehicle Creation Button, once all Vehicle scenarios of step 3 are done, initializes the other variables needed for the program to work correctly, positions the button appropriately and finally shows it.

5. Nothing happens until the user clicks the Vehicle Creation Button.

6. When the user clicks the Vehicle Creation Button, the handling scenario hides the button, sets the number of vehicles waiting to be served to 3, and then calls the DefineRandomVehicleServiceOrder procedure which determines the random sequence the clones of the three vehicles that will run in this round will be created in.
The scenario then waits for all vehicles to be served and subsequently reshows the Vehicle Creation Button.

Program Control Flow

7. When a clone of any of the three vehicles starts, it executes in sequence the following procedures:

- **It shows the clone at the start of the corresponding lane and then moves it toward the pump, i.e., until it reaches the yellow color of the lane (MoveTowardThePump procedure).**

- **It waits for the semaphore to become 1 and then decrements it to 0, reserving this way the pump (ReservePump procedure).**

- **It then executes the transaction with the pump: if the Vehicle's reservoirCapacity is less than or equal to the tankFuel it then notifies the user that there is enough fuel in the tank for this transaction to complete,and then decrements the tankFuel by the Vehicle's reservoirCapacity. Otherwise, it just notifies the user that there is not enough fuel in the tank and therefore the transaction will be cancelled (ExecuteTransaction procedure).**

- **Then follows the ReleasePump procedure that simply sets the semaphore back to 1.**

- **Finally, there is the Depart procedure that moves the Vehicle to the right, and when that touches the right side of the Scratch stage the procedure decrements the vehiclesAwaitingToBeServed by 1 and deletes the current clone.**

Program Control Flow

Vehicle

When the user clicks the green flag:

- it hides,
- positions itself at the start of the corresponding lane,
- andsets up its attributes (a random color for its appearance, and a random reservoirCapacity from the set {10, 20, 30, 40}).

When it receives the "Count Vehicles" message:

- it increments the numberOfVehicles count by 1.

When the Vehicle clone starts:

- It shows the clone at the start of the corresponding lane and then moves it toward the pump, i.e., until it reaches the yellow color of the lane (MoveTowardThePump procedure).

- It waits for the semaphore to become 1 and then decrements it to 0, thus reserving the pump (ReservePump procedure).

- It then executes the transaction with the pump:

  If the Vehicle's reservoirCapacity is less than or equal to the tankFuel,

  then

  it notifies the user that there is enough fuel in the tank for this transaction to complete,

  and then decrements the tankFuel by the Vehicle's reservoirCapacity

  otherwise,

  it just notifies the user that there is not enough fuel in the tank and therefore the transaction will be cancelled (ExecuteTransaction procedure).

- Then it executes the ReleasePump procedure that simply sets the semaphore back to 1.
- Finally, it runs the Depart procedure that moves the Vehicle to the right, and when that touches the right side of the Scratch stage the procedure decrements the vehiclesAwaitingToBeServed by 1 and deletes the current clone.

Program Control Flow

Vehicle Creation Button

When the user clicks the green flag:

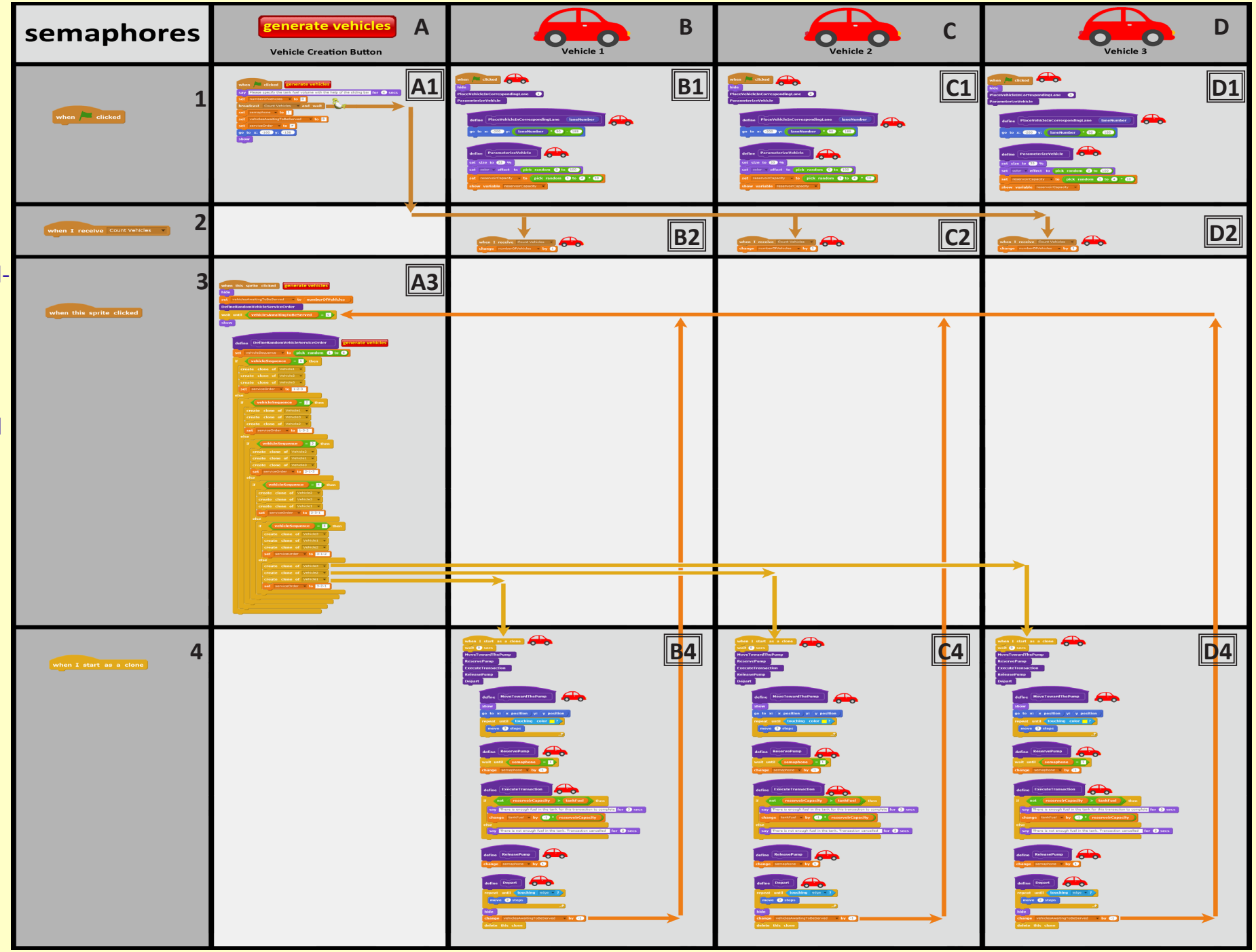
- It notifies the user that they must specify the tank fuel volume (by using the sliding bar under the tankFuel variable),
- initializes the numberOfVehicles counter to 0,
- broadcasts the message "Count Vehicles", waiting for all interested sprites to handle it,
- initializes the other variables needed for the program to work correctly,
- positions itself appropriately, and finally
- shows.

When the user clicks the Vehicle Creation Button:

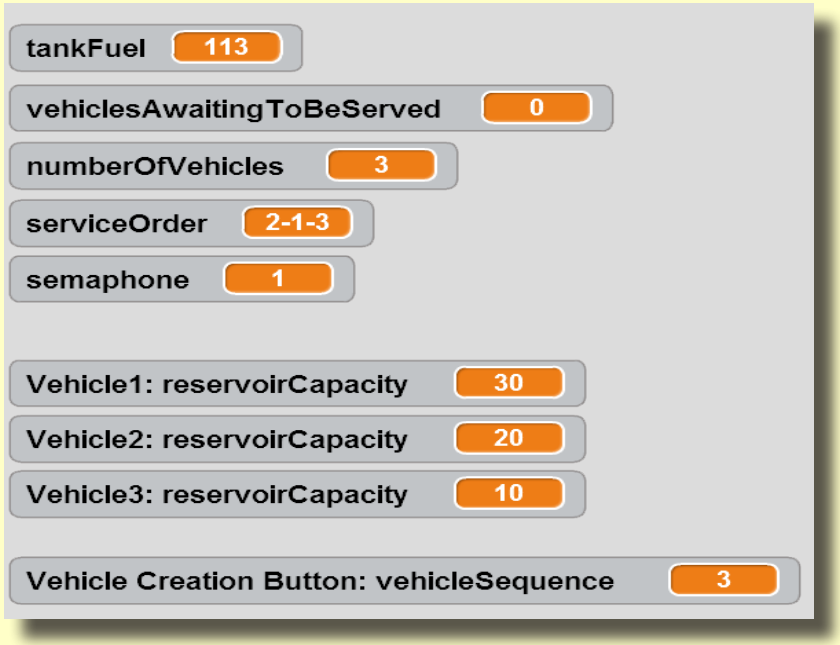
- it hides the button,
- sets the number of vehicles waiting to be served to 3,
- calls the DefineRandomVehicleServiceOrder procedure which determines the random sequence the clones of the three vehicles that will run in this round will be created in,
- waits for all vehicles to be served and subsequently reshows the Vehicle Creation Button.

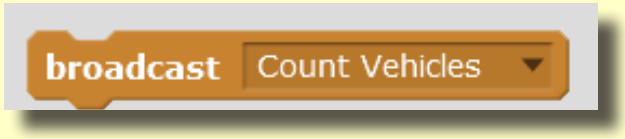
CodOrama

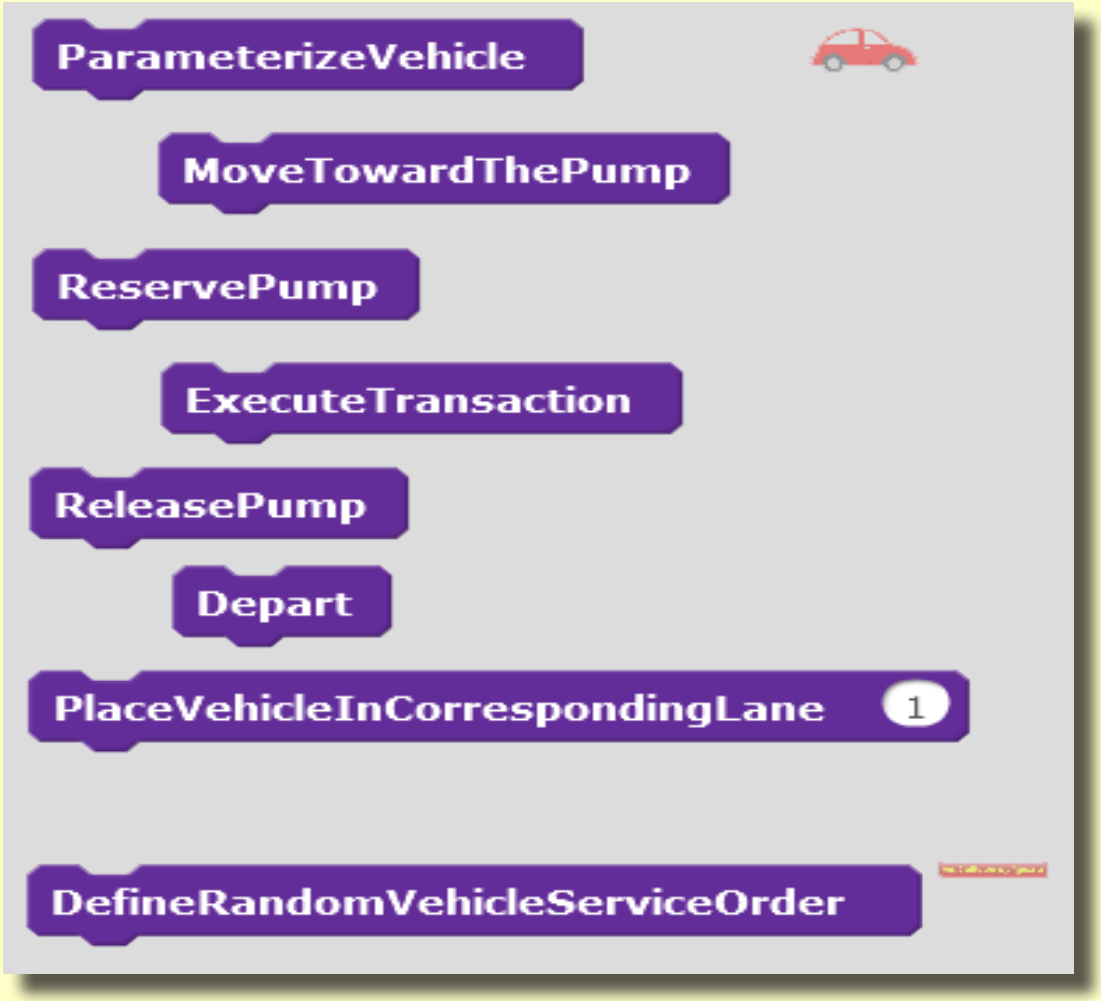
The table on this page, where we can see the various code modules and their interconnections, may be used in order to understand these scenarios.



Variables - messages - procedures



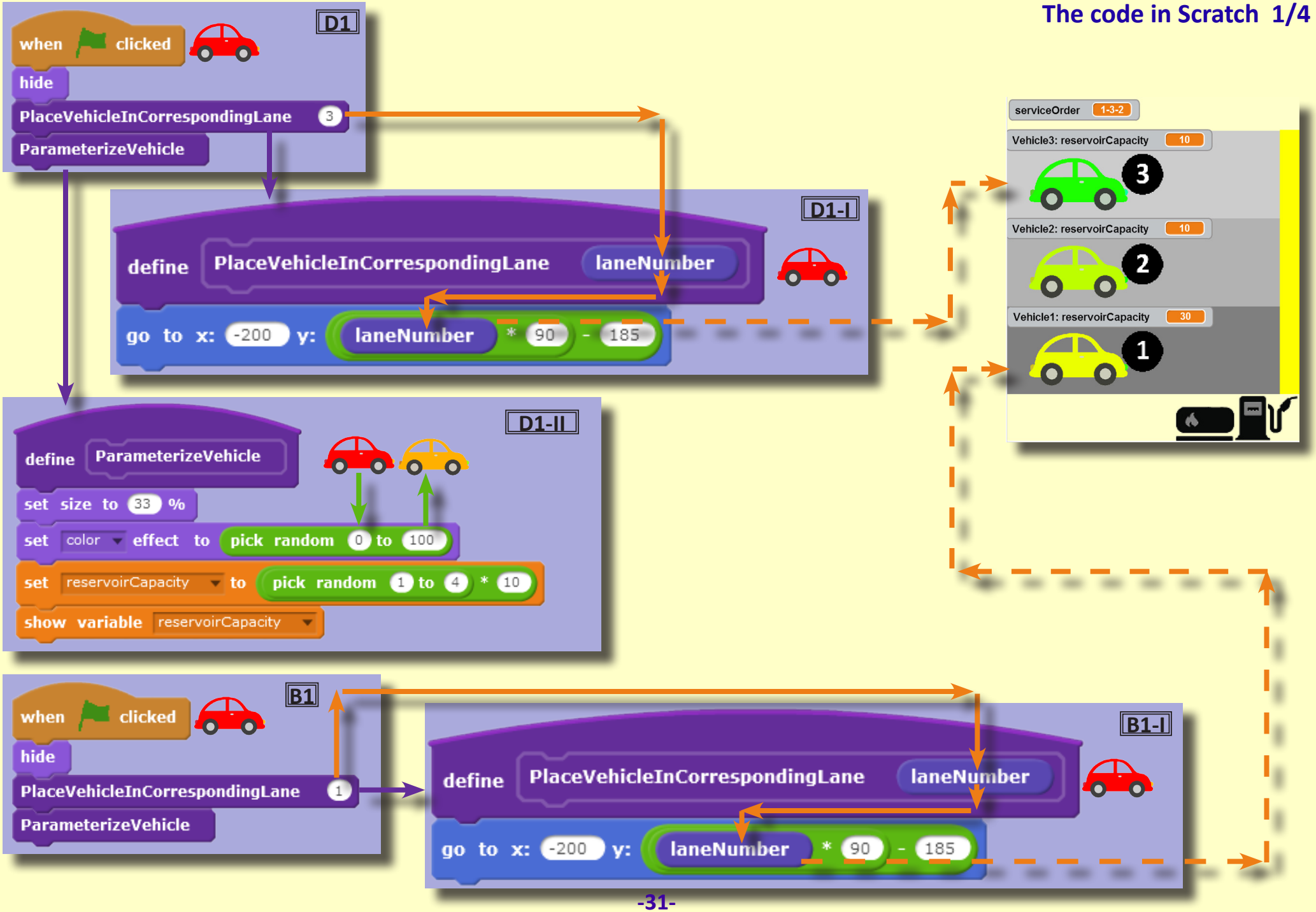


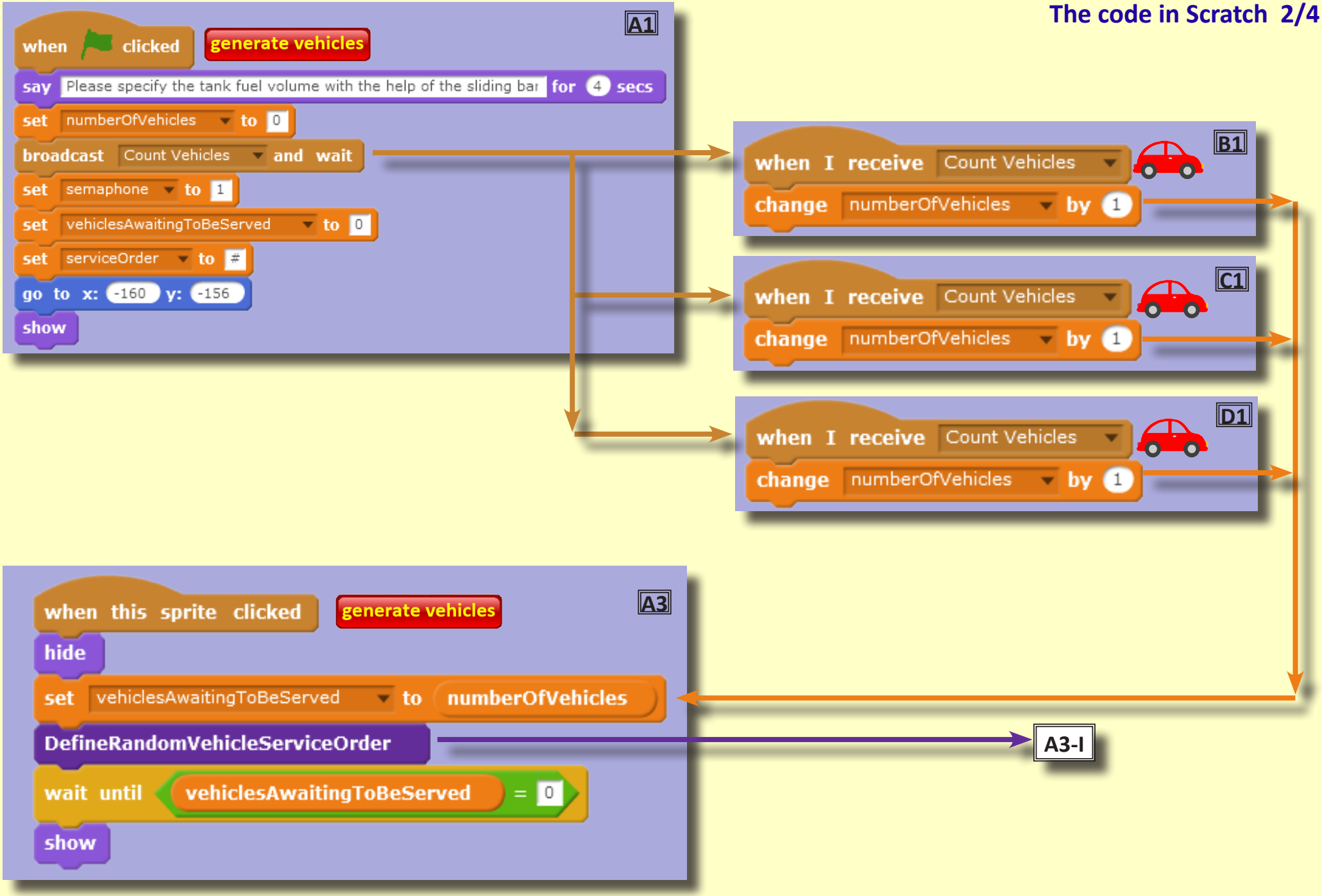


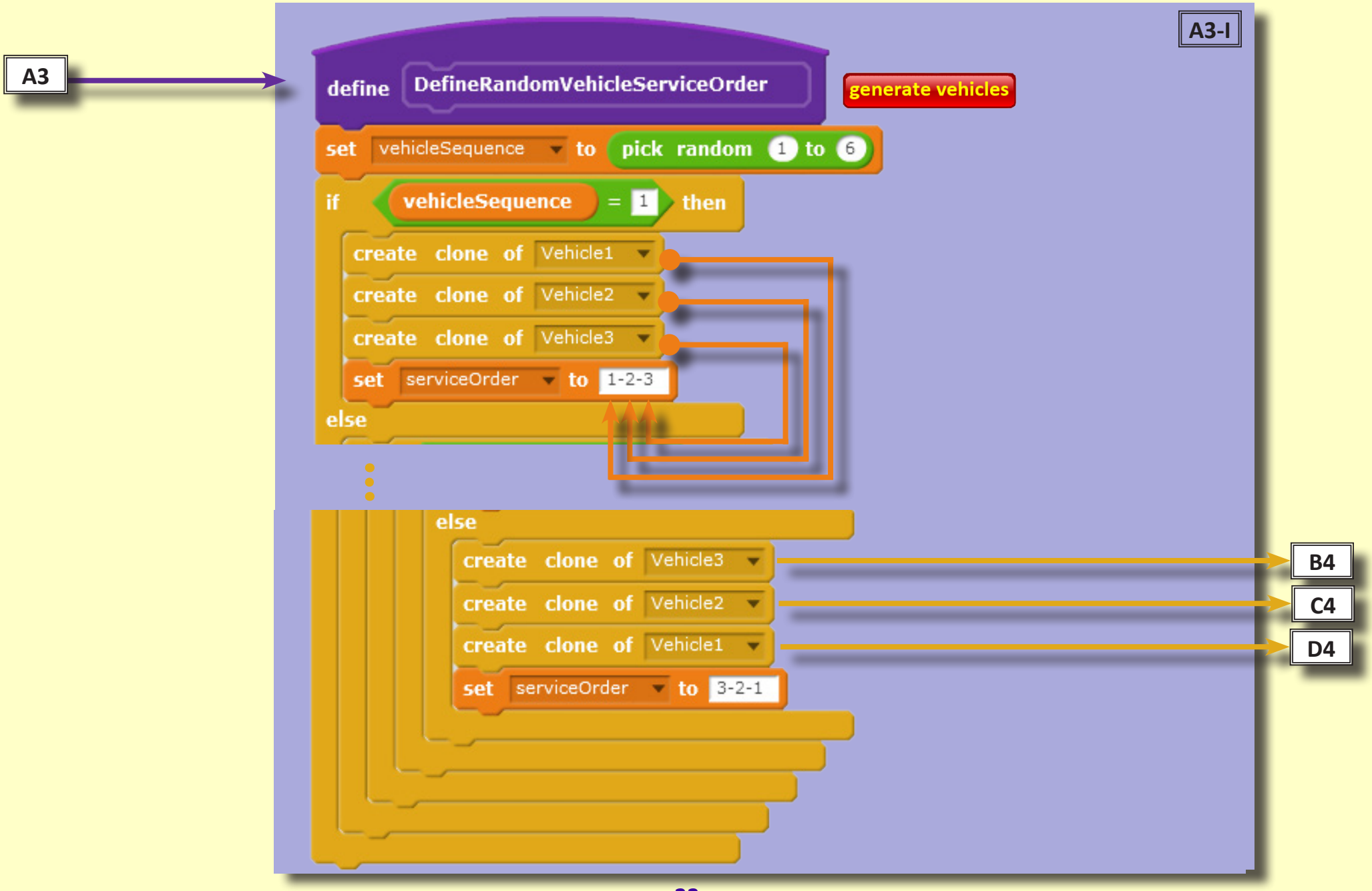
The code in Scratch

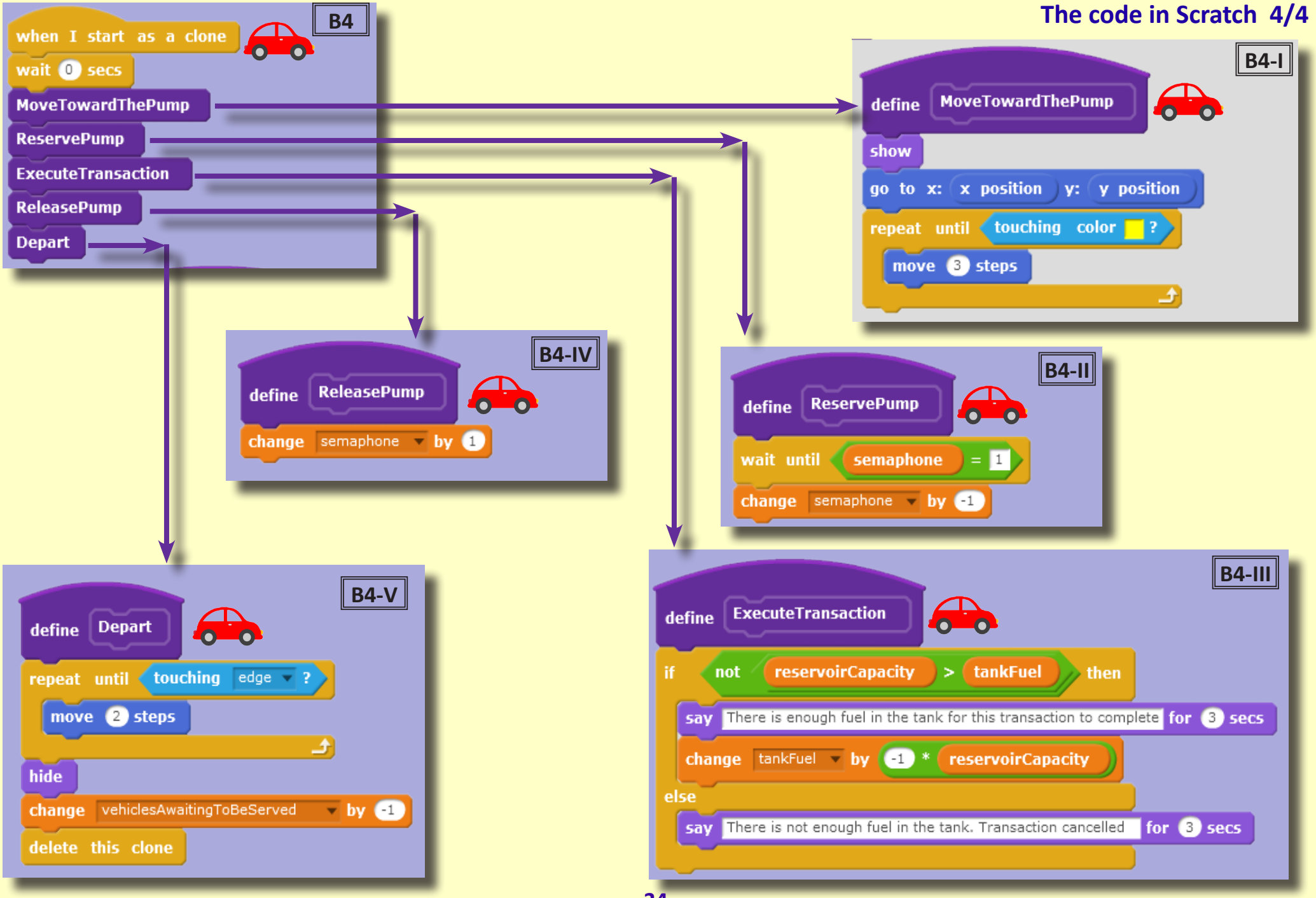
CodOrama: https://prezi.com/ikw6obvr73f6/semaphore/ or http://bit.ly/1X7CnCk

Code: https://scratch.mit.edu/projects/82693844/ or http://bit.ly/1jEzXwg

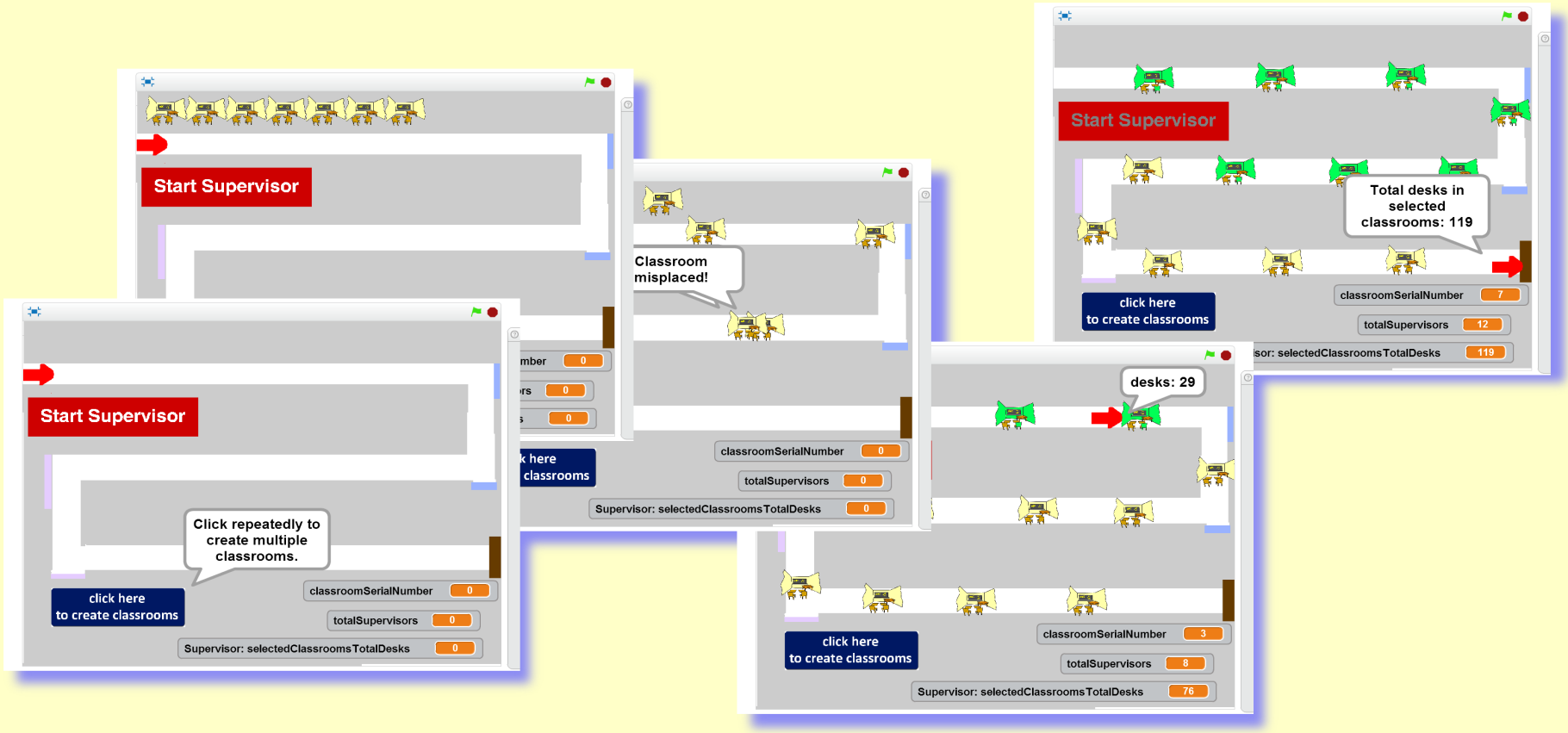

D1
when clicked
hide
PlaceVehicleInCorrespondingLane 3
ParameterizeVehicle
D1-I
define PlaceVehicleInCorrespondingLane laneNumber
go to x: -200 y: laneNumber * 90 - 185
D1-II
define ParameterizeVehicle
set size to 33 %
set color effect to pick random 0 to 100
set reservoirCapacity to pick random 1 to 4 * 10
show variable reservoirCapacity
serviceOrder 1-3-2
Vehicle3: reservoirCapacity 10
3
Vehicle2: reservoirCapacity 10
2
Vehicle1: reservoirCapacity 30
1
B1
when clicked
hide
PlaceVehicleInCorrespondingLane 1
ParameterizeVehicle
B1-I
define PlaceVehicleInCorrespondingLane laneNumber
go to x: -200 y: laneNumber * 90 - 185


A1
when clicked
generate vehicles
say Please specify the tank fuel volume with the help of the sliding bar for 4 secs
set numberOfVehicles to 0
broadcast Count Vehicles and wait
set semaphone to 1
set vehiclesAwaitingToBeServed to 0
set serviceOrder to #
go to x: -160 y: -156
show
B1
when I receive Count Vehicles
change numberOfVehicles by 1
C1
when I receive Count Vehicles
change numberOfVehicles by 1
D1
when I receive Count Vehicles
change numberOfVehicles by 1
A3
when this sprite clicked
generate vehicles
hide
set vehiclesAwaitingToBeServed to numberOfVehicles
DefineRandomVehicleServiceOrder
A3-I
wait until vehiclesAwaitingToBeServed = 0
show

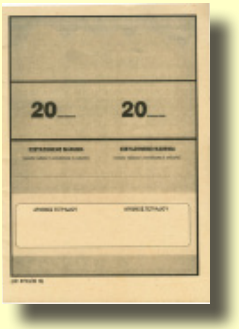

A3
A3-I
define DefineRandomVehicleServiceOrder
generate vehicles
set vehicleSequence to pick random 1 to 6
if vehicleSequence = 1 then
create clone of Vehicle1
create clone of Vehicle2
create clone of Vehicle3
set serviceOrder to 1-2-3
else
else
create clone of Vehicle3
create clone of Vehicle2
create clone of Vehicle1
set serviceOrder to 3-2-1
B4
C4
D4


B4
when I start as a clone
wait 0 secs
MoveTowardThePump
ReservePump
ExecuteTransaction
ReleasePump
Depart
B4-I
define MoveTowardThePump
show
go to x: x position y: y position
repeat until touching color ?
move 3 steps
B4-IV
define ReleasePump
change semaphone by 1
B4-II
define ReservePump
wait until semaphone = 1
change semaphone by -1
B4-V
define Depart
repeat until touching edge ?
move 2 steps
hide
change vehiclesAwaitingToBeServed by -1
delete this clone
B4-III
define ExecuteTransaction
if not reservoirCapacity > tankFuel then
say There is enough fuel in the tank for this transaction to complete for 3 secs
change tankFuel by -1 * reservoirCapacity
else
say There is not enough fuel in the tank. Transaction cancelled for 3 secs

3. Exam center supervisor



Hellenic National Examinations Question



SECONDARY SCHOOL LEAVING EXAMINATION
MAY 30TH 2006
TESTED COURSE: DEVELOPING APPLICATIONS IN A PROGRAMMING ENVIRONMENT

Question 3

In a state examination there are 1,500 candidates. A building with classrooms of various capacities is being used as the exam venue. The number of supervisors needed per classroom is specified according to the following rules:

Classroom capacity	Number of supervisors
Up to 15 desks	1
16 – 23 desks	2
More than 23 desks	3

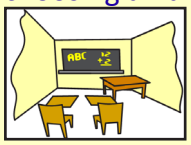
Write a program in the "Language" programming language which:

a. Will read the capacity of each classroom, and then will calculate and show the number of required supervisors. The calculation of the number of supervisors is to be implemented by a function you must write. (12 points)

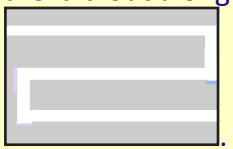
b. Will stop executing when the total number of exam desks needed has been reached. (8 points)

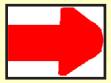
Note: Assume the total classroom capacity of the venue is adequate for the number of candidate.

Hellenic National Examinations Question - Tested course: Developing Applications in a Programming Environment - Scratch adaptation

The Supervisor of a school complex has been assigned the task of choosing a number of Classrooms  that will serve as an exam venue for 100 examinees.

The Classrooms in the complex are laid out along a corridor.



The Supervisor  enters the corridor at the top left corner and exits at the bottom right.

The number of desks per Classroom are in the range of 1-30.

A number of exam supervisors per Classroom is to be determined depending Classroom is to be determined depending on the number of its desks according to the following rule:

Classroom desks	Number of supervisors
1-15	1
16-23	2
24-30	3

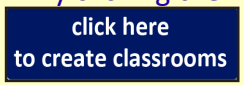
Using the “half-baked” Scratch program (where you can find the fully implemented sprites and scenarios of the problem’s non-core agents), you are to write the Supervisor’s scenarios with which he/she will choose in sequence the Classrooms that are necessary for the exam to take place.

a) For each selected Classroom the algorithm must show the Classroom’s sequence number, the number of its desks, and the number of its supervisors.

b) When the Supervisor reaches the exit, they must show the total number of desks over all selected Classrooms, as well as the total number of required classroom supervisors.

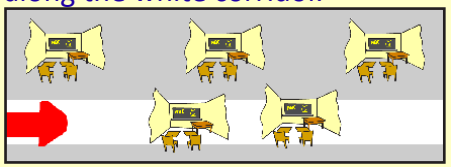
How to use the program

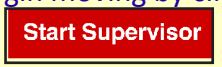
1. Click the green flag. 

2. By clicking the 

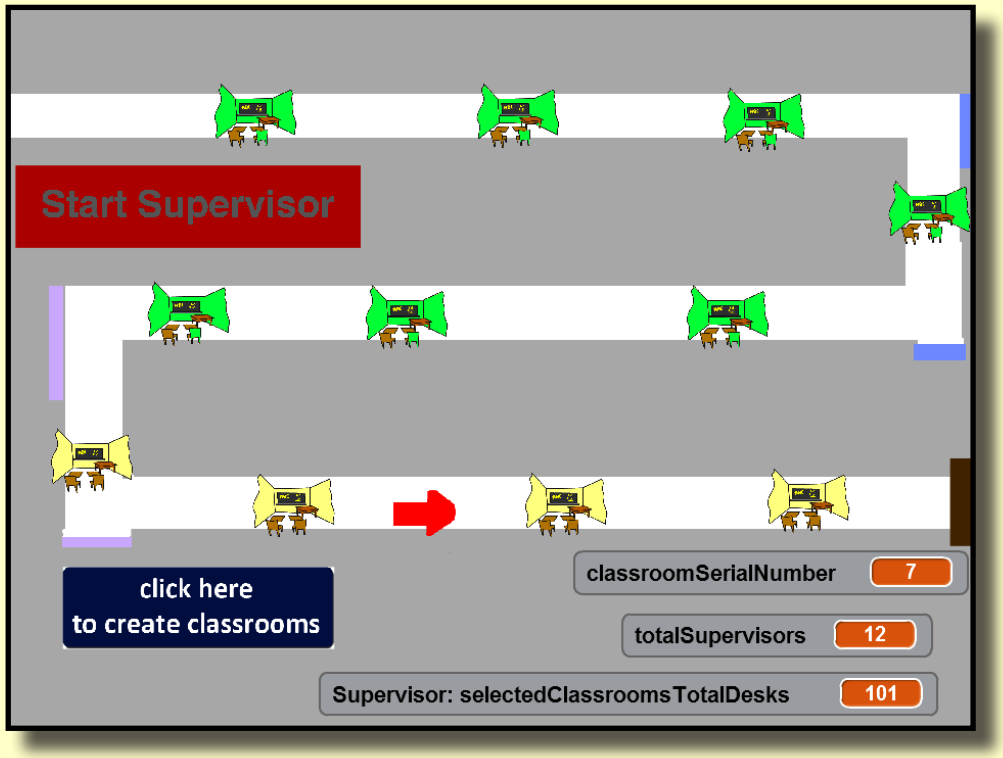
button you can create new Classrooms (up to 20).

3. Drag and drop as many Classrooms as you wish along the white corridor.



4. Ask the Supervisor to begin moving by clicking the 

button



Program Control Flow

1. When the user clicks the green flag, the Classroom Creation Button positions itself at the right place, initializes the number of classrooms to 0, and finally displays to the user instructions on how to create and place classrooms in the corridor.

2. When the user clicks theClassroom Creation Button, if the numberOfClassrooms that have already been created is up to 18, the corresponding scenario increments the number of classrooms by one and then creates a new clone of the Classroom sprite. Moreover, it sets up the appearance of the button to instantlyreflect its clicking by the user.

3. When a new clone of the Classroom sprite is being created:

a. The ClassroomInitialOperations procedure places the clone at the next free position above the corridor (by using the classroomInitialXPos variable), notifies the program sprites that classroomsAreProperlyPlaced (since there's no overlapping between them), and finally switches to its yellow costume.

b. Subsequently, the CheckIfClassroomOverlapsAnother procedure enters an infinite loop in order to check repeatedly whether this clone touches the yellow color of another Classroom clone. If this is so, then the code of this procedure resets the classroomsAreProperlyPlaced flag variable to 0 and notifies the user that they have misplaced this clone, so that the user can subsequently reposition it away from the other overlapped clone. Otherwise (if this clone is not touching a different Classroom clone), the code raises the classroomsAreProperlyPlaced flag to 1.

Program Control Flow

4. When the user clicks the green flag, the Classroom places itself in the appropriate position, hides itself and then initializes the classroomInitialXPos variable for the first clone to be createdlater on. This variable specifies the horizontal position which every new Classroom clone created by the user will be placed in (see step 3a above).

5. When the user clicks the green flag, the Start Button places itself in the appropriate position, switches to the Enabled button costume and lowers the startSupervisorButtonClicked flag variable to 0.

6. When the user clicks the green flag, the Exit sprite places itself to the appropriate position.

7. When the user clicks the green flag, the Supervisor sprite initializes all relevant variables (InitializeVariables procedure) and sets up its appearance (InitializeSupervisorAppearance procedure). At the end of this procedure it clears the flag variable begin to 0 and then waits for it to be raised (to become 1).

8. When the user clicks the Start Button, the begin flag is raised (to 1). More specifically, the handling scenario, after having checked that classroomsAreProperlyPlaced (i.e., there are no overlapping classrooms in the corridor), disables the button and raises the begin flag (to 1). Otherwise (if any classrooms have been misplaced), it clears the begin flag. All of the above in this step is done only once – the first time this button is clicked. After that, we make sure the user cannot click again the button by manipulating the startSupervisorButtonClicked variable.

Program Control Flow

9. As soon as the begin flag is raised to 1, the main Supervisor scenario waiting on it (see step 7 above) continues running and enters a loop that is repeated until the Supervisor reaches the end of the corridor (i.e., until it touches the Exit). In that loop (in each iteration of it):

a. It calls the MoveSupervisor procedure that takes care of moving the Supervisor in the corridor a few steps each time through.

b. It checks whether the Supervisor touches the yellow color of any Classroom clone that it may encounter in the corridor and in that case calls the CalculateDeskAndSupervisorTotals procedure.

c. The CalculateDeskAndSupervisorTotals procedure checks in turn whether the sum total of desks across all selected by the Supervisor classrooms (selectedClassroomsTotalDesks) are less than 100 and if so, it broadcasts the message "Classroom is being selected" and then waits until the scenario handling that message is done.

Program Control Flow

11. Upon completion of the OnClassroomSelectedprocedure (in the Classroom clones' scenario that is handling the message "Classroom is being selected"), program control returns to the Supervisor procedure CalculateDeskAndSupervisorTotals, which then:

a. Increments the classroomSerialNumber by 1.

b. Increments the selectedClassroomsTotalDesks by the current classroom's number of desks (classroomDesks).

c. Calls the CalculateClassroomSupervisors procedure, which calculates the required supervisors for the current classroom based on the rule of the table shown in page 37.

d. Increments the totalSupervisors by the current classroomSupervisors.

e. Calls procedure ShowMessagesForSelectedClassroom that shows the user the information about the desks and the required supervisors for the classroom that has just been selected.

12. Beyond the Supervisor basic scenario's loop (i.e., as soon as the Supervisor touches the Exit), the scenario calls the FlashSupervisor procedure serving as a visual cue to the user that the Supervisor has reached the end of the corridor.

13. Finally, it calls the Supervisor's ShowFinalMessages procedure to notify the user of the total number of desks and the total number of required supervisors for the selected classrooms.

Sprites' Functional Analysis

Start Button

When the user clicks the green flag:

- The Start Button places itself in the appropriate position,
- switches to the Enabled button costume,
- andclears the startSupervisorButtonClicked flag variable to 0.

When the user clicks the Start Button:

- If the Start Button has not been previously clicked (i.e., if the startSupervisorButtonClicked flag is 0)

  then if classroomsAreProperlyPlaced

  then

  - the startSupervisorButtonClicked flag is raised to 1,
  - the button's costume is changed to Disabled button,
  - the begin flag is raised (to 1)

  otherwise

  - the begin flag is cleared (to 0).

Exit

When the user clicks the green flag:

- the Exit sprite places itself in the appropriate position.

Classroom Creation Button

When the user clicks the green flag:

- The button places itself in the appropriate position,
- initializes the numberOfClassrooms to 0, and
- displays to the user instructions on how to create and place classrooms in the corridor.

When the user clicks the Classroom Creation Button:

- if the numberOfClassrooms that have already been created is up to 18,

  then

  - the numberOfClassrooms is incremented by one,
  - a new clone of the Classroom sprite is being created,
  - the appearance of the button is being set up to instantly reflect its clicking by the user.

Sprites' Functional Analysis

Classroom

When the user clicks the green flag:

- **The Classroomplaces itself in the appropriate position,**
- **Hides itself,**
- **and then initializes the classroomInitialXPos variable for the first clone to be createdlater on. This variable specifies the horizontal position which every new Classroom clone created by the user will be placed in.**

When a new clone of the Classroom sprite is being created:

a) The ClassroomInitialOperations procedure:

- **places the new clone in the next free position above the corridor (by using the classroomInitialXPos variable),**
- **notifies the program sprites that classroomsAreProperlyPlaced (since there's no overlapping between them),**
- **switches to its yellow costume.**

b) The CheckIfClassroomOverlapsAnother procedure:

- **enters an infinite loop in order to check repeatedly:**

if this clone touches the yellow color of another Classroom clone

then

- **the classroomsAreProperlyPlaced flag variable is reset to 0,**
- **the user is being notified that they have misplaced this clone, so that they can subsequently reposition it away from the other overlapped clone.**

otherwise (if this clone is not touching a different Classroom clone),

- **the classroomsAreProperlyPlaced flag is raised to 1.**

When The message "Classroom is being selected" is being received:

- **the OnClassroomSelected procedure is called, which:**

if the clone is touching the Supervisor

then

- **o the number of classroom desks is assigned a random value between 1 and 30**
- **o the clone's color changes from yellow to green (by switching the sprite costume to the green one).**

Sprites' Functional Analysis

Supervisor

When the user clicks the green flag:

- initializes all relevant variables (InitializeVariables procedure),
- sets up its appearance (InitializeSupervisorAppearance procedure).

  At the end of this procedure
  - it clears the flag variable begin to 0
  - and then waits for it (begin) to be raised (to become 1).

As soon as the begin flag is raised to 1, it continues running and enters a loop that is repeated until the Supervisor reaches the end of the corridor (i.e., until it touches the Exit). In that loop (in each iteration of it):

- It calls the MoveSupervisor procedure that moves the Supervisor in the corridor a few steps each time through.
- It checks whether the Supervisor touches the yellow color of any Classroom clone that it may encounter in the corridor and in that case calls the CalculateDeskAndSupervisorTotals procedure. In that procedure:

  If the sum total of desks across all selected by the Supervisor classrooms (selected ClassroomsTotalDesks) are less than 100

  then
  - it broadcasts the message "Classroom is being selected"
  - and waits until the scenario of all the Classroom clones handling that message is done.

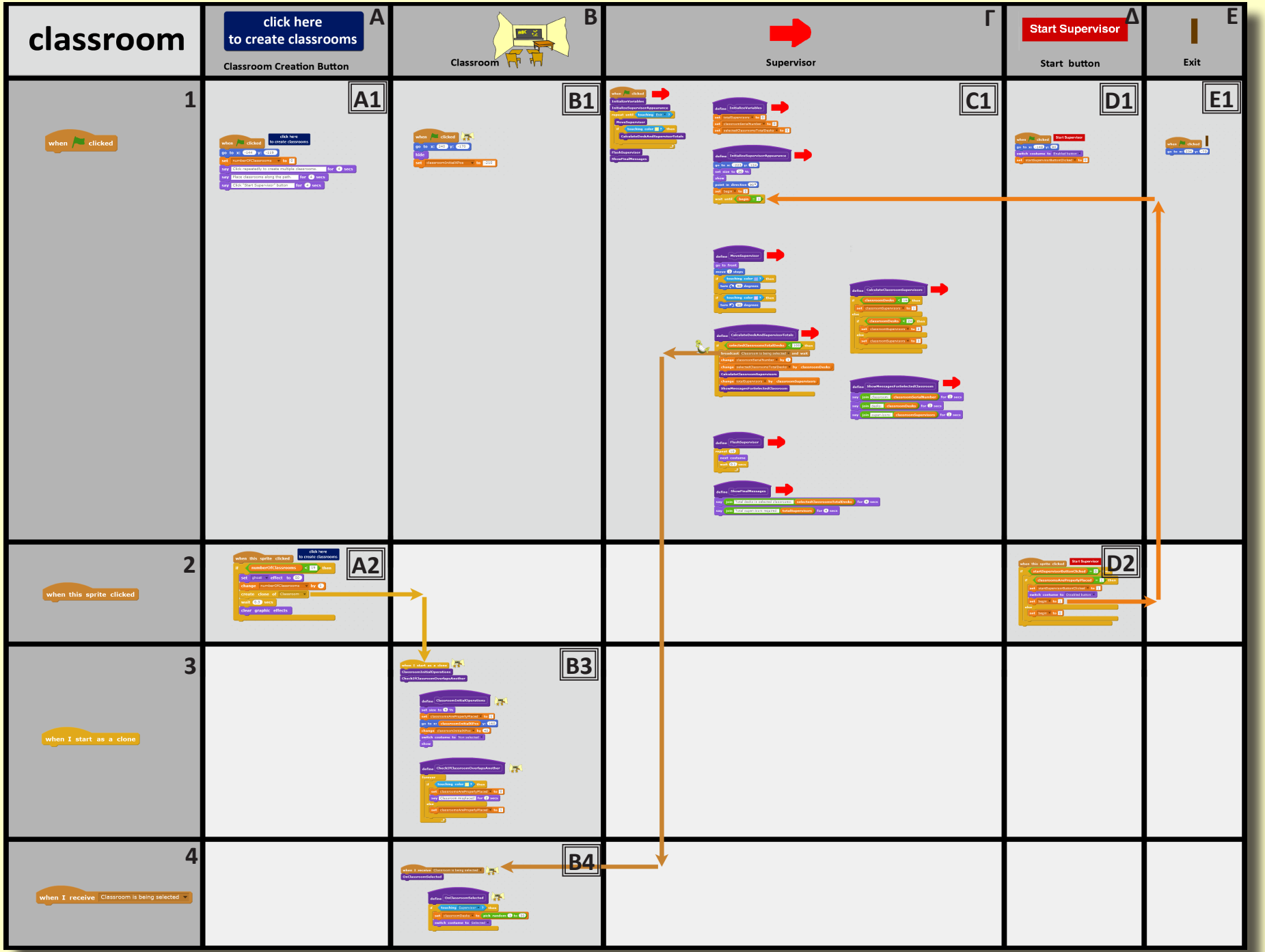
  Upon completion of the Classroom clones' scenario handling the message "Classroom is being selected", CalculateDeskAndSupervisorTotalsgoes on to:
  - Increment the classroomSerialNumber by 1.
  - Increment the selectedClassroomsTotalDesks by the current classroom's number of desks (classroomDesks).
  - Call the CalculateClassroomSupervisors procedure, which calculates the required supervisors for the current classroom based on the rule of the table.
  - Increment the totalSupervisors by the current classroomSupervisors.
  - Call procedure ShowMessagesForSelectedClassroom that shows the user the information about the desks and the required supervisors for the classroom that has just been selected.

Classroom desks	Number of supervisors
1-15	1
16-23	2
24-30	3

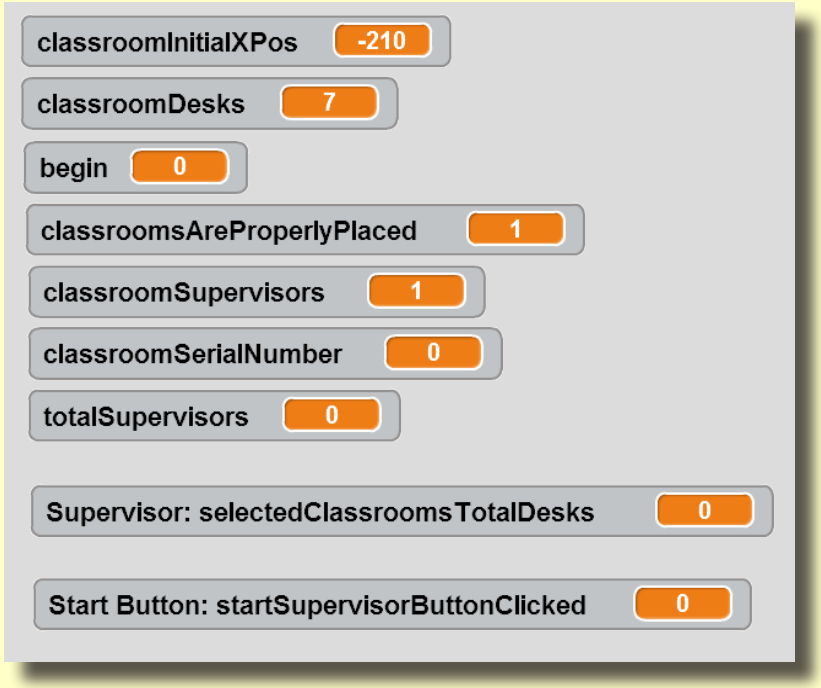
- Beyond the Supervisor basic scenario's loop (i.e., as soon as the Supervisor touches the Exit), the scenario calls the FlashSupervisor procedure to indicate that the Supervisor has reached the end of the corridor.
- Finally, it calls the ShowFinalMessages procedure to notify the user of the total number of desks and the total number of required supervisors for the selected classrooms.

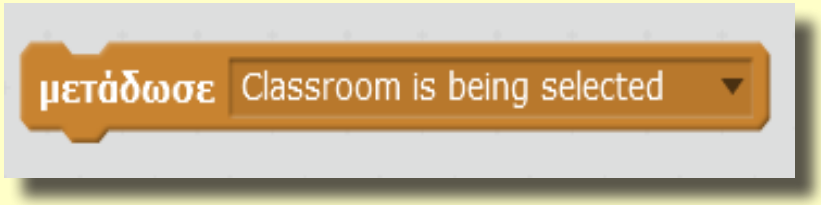
CodOrama

The table on this page, where we can see the various code modules and their interconnections, may be used in order to understand these scenarios.



Μεταβλητές - μηνύματα - διαδικασίες

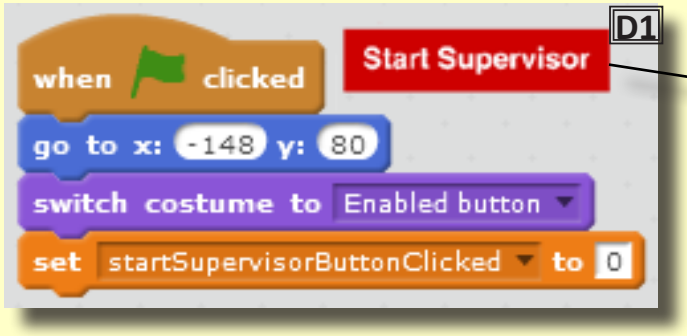


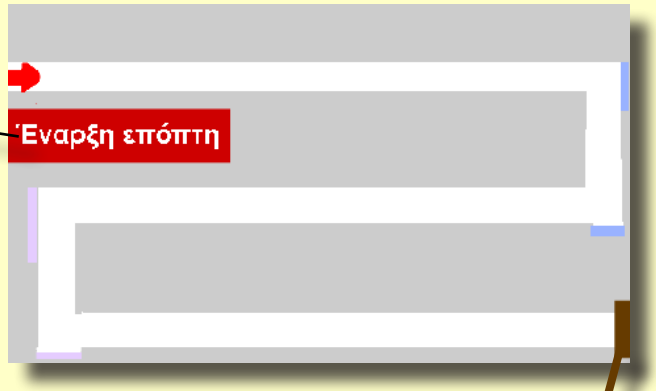


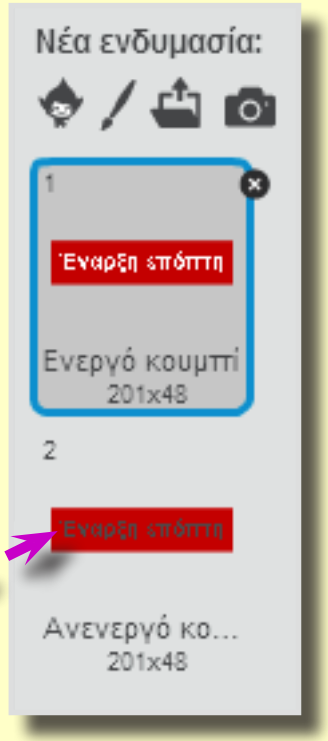
The code in Scratch

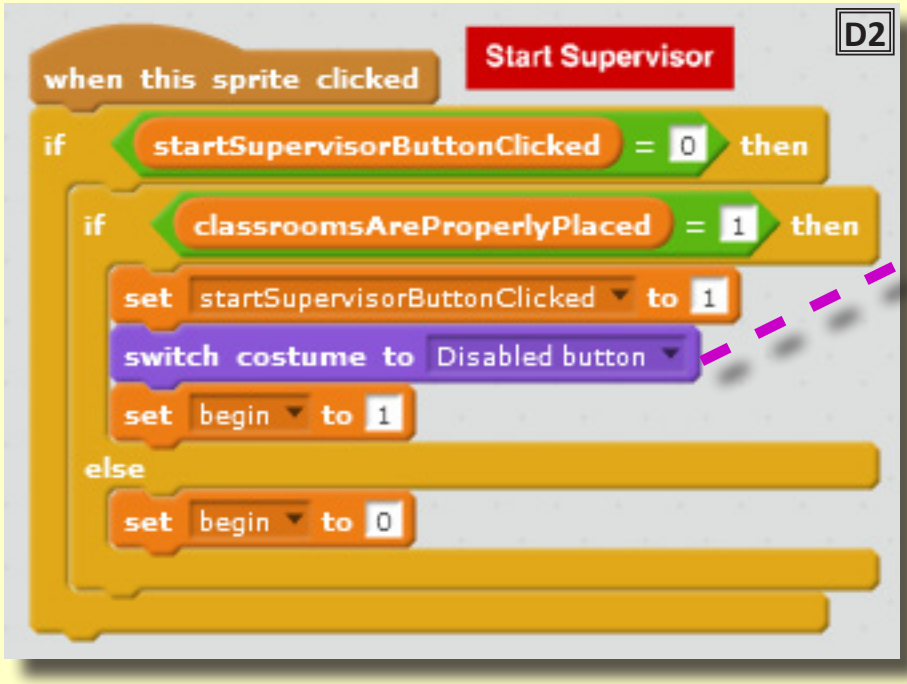
CodOrama: https://prezi.com/xzycf83eyech/exam-center-supervisor/ or http://bit.ly/1OxNiDJ

Code: https://scratch.mit.edu/projects/82696432/ or http://bit.ly/1OxMMpl

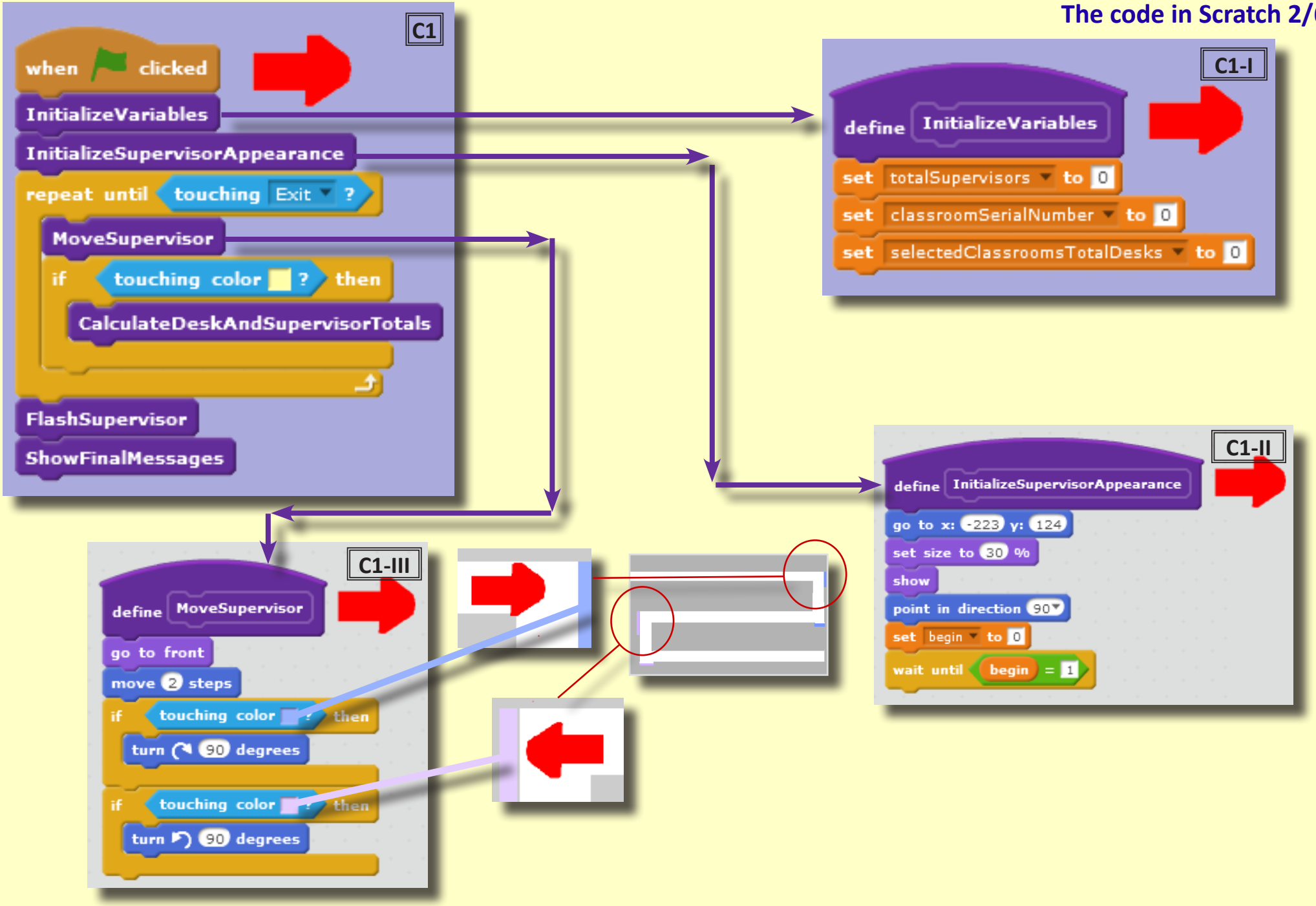

D1
when clicked
Start Supervisor
go to x: -148 y: 80
switch costume to Enabled button
set startSupervisorButtonClicked to 0


Έναρξη επόπτη


Νέα ενδυμασία:
1
Έναρξη επόπτη
Ενεργό κουμπί
201x48
2
Έναρξη επόπτη
Ανενεργό κο...
201x48

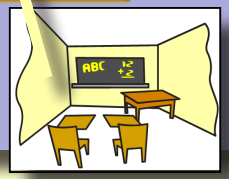

D2
when this sprite clicked
Start Supervisor
if startSupervisorButtonClicked = 0 then
if classroomsAreProperlyPlaced = 1 then
set startSupervisorButtonClicked to 1
switch costume to Disabled button
set begin to 1
else
set begin to 0


E1
when clicked
go to x: 236 y: -70


C1
when clicked
InitializeVariables
InitializeSupervisorAppearance
repeat until touching Exit ?
MoveSupervisor
if touching color ? then
CalculateDeskAndSupervisorTotals
FlashSupervisor
ShowFinalMessages
C1-I
define InitializeVariables
set totalSupervisors to 0
set classroomSerialNumber to 0
set selectedClassroomsTotalDesks to 0
C1-II
define InitializeSupervisorAppearance
go to x: -223 y: 124
set size to 30 %
show
point in direction 90
set begin to 0
wait until begin = 1
C1-III
define MoveSupervisor
go to front
move 2 steps
if touching color then
turn 90 degrees
if touching color then
turn 90 degrees

C1

when clicked
InitializeVariables
InitializeSupervisorAppearance
repeat until touching Exit ?
MoveSupervisor
if touching color ? then
CalculateDeskAndSupervisorTotals
FlashSupervisor
ShowFinalMessages



C1-IV

define CalculateDeskAndSupervisorTotals
if selectedClassroomsTotalDesks < 100 then
broadcast Classroom is being selected and wait
change classroomSerialNumber by 1
change selectedClassroomsTotalDesks by classroomDesks
CalculateClassroomSupervisors
change totalSupervisors by classroomSupervisors
ShowMessagesForSelectedClassroom

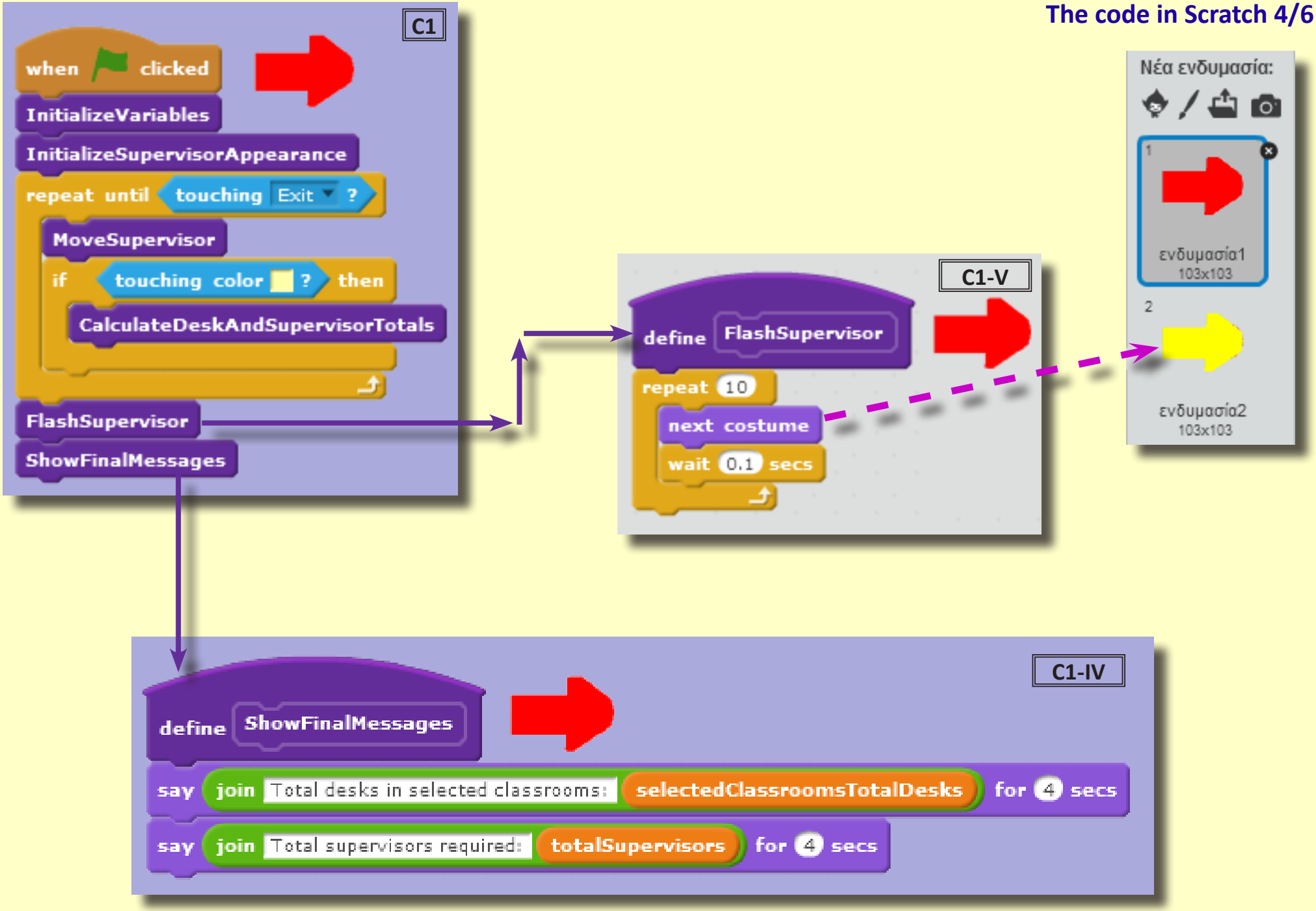
B4

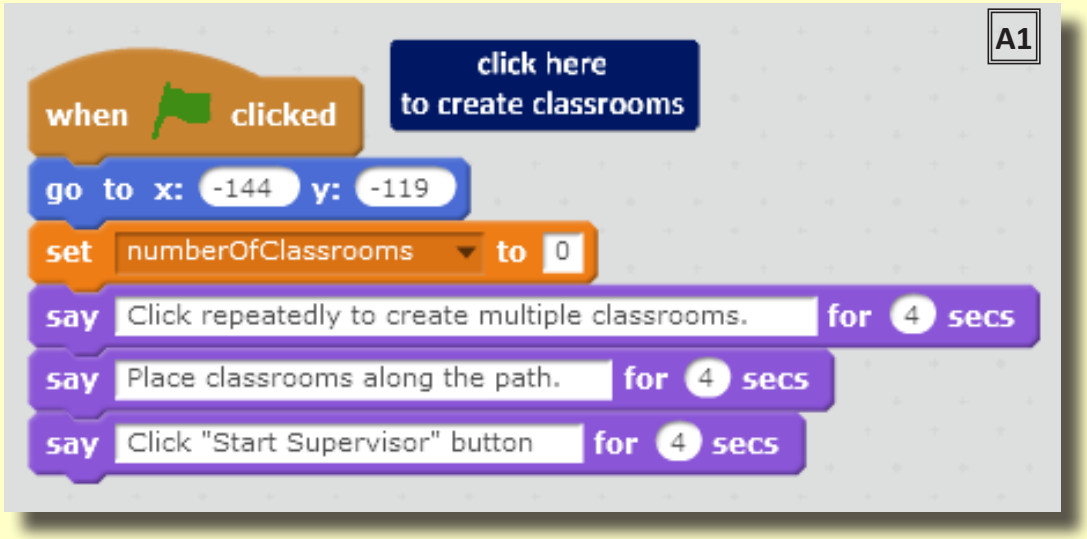
C1-IV-b

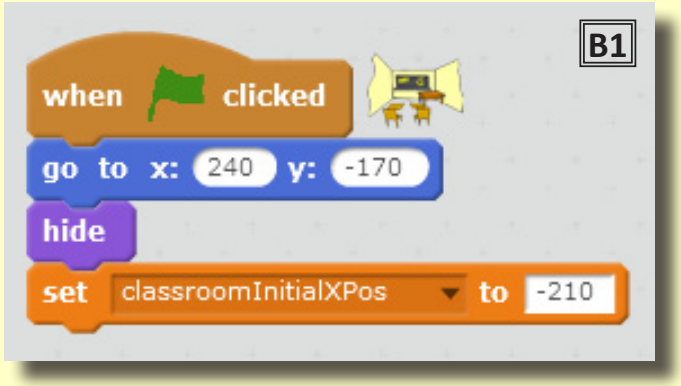
define ShowMessagesForSelectedClassroom
say join classroom: classroomSerialNumber for 2 secs
say join desks: classroomDesks for 2 secs
say join supervisors: classroomSupervisors for 2 secs

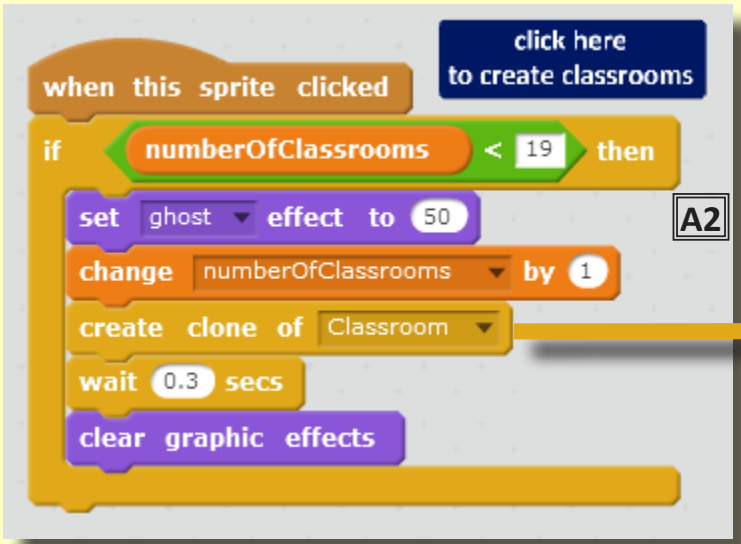
C1-IV-a

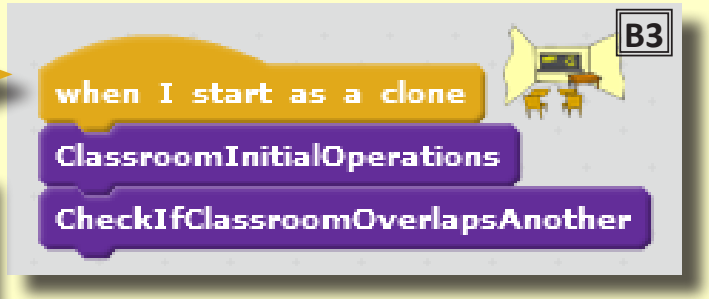
define CalculateClassroomSupervisors
if classroomDesks < 16 then
set classroomSupervisors to 1
else
if classroomDesks < 24 then
set classroomSupervisors to 2
else
set classroomSupervisors to 3

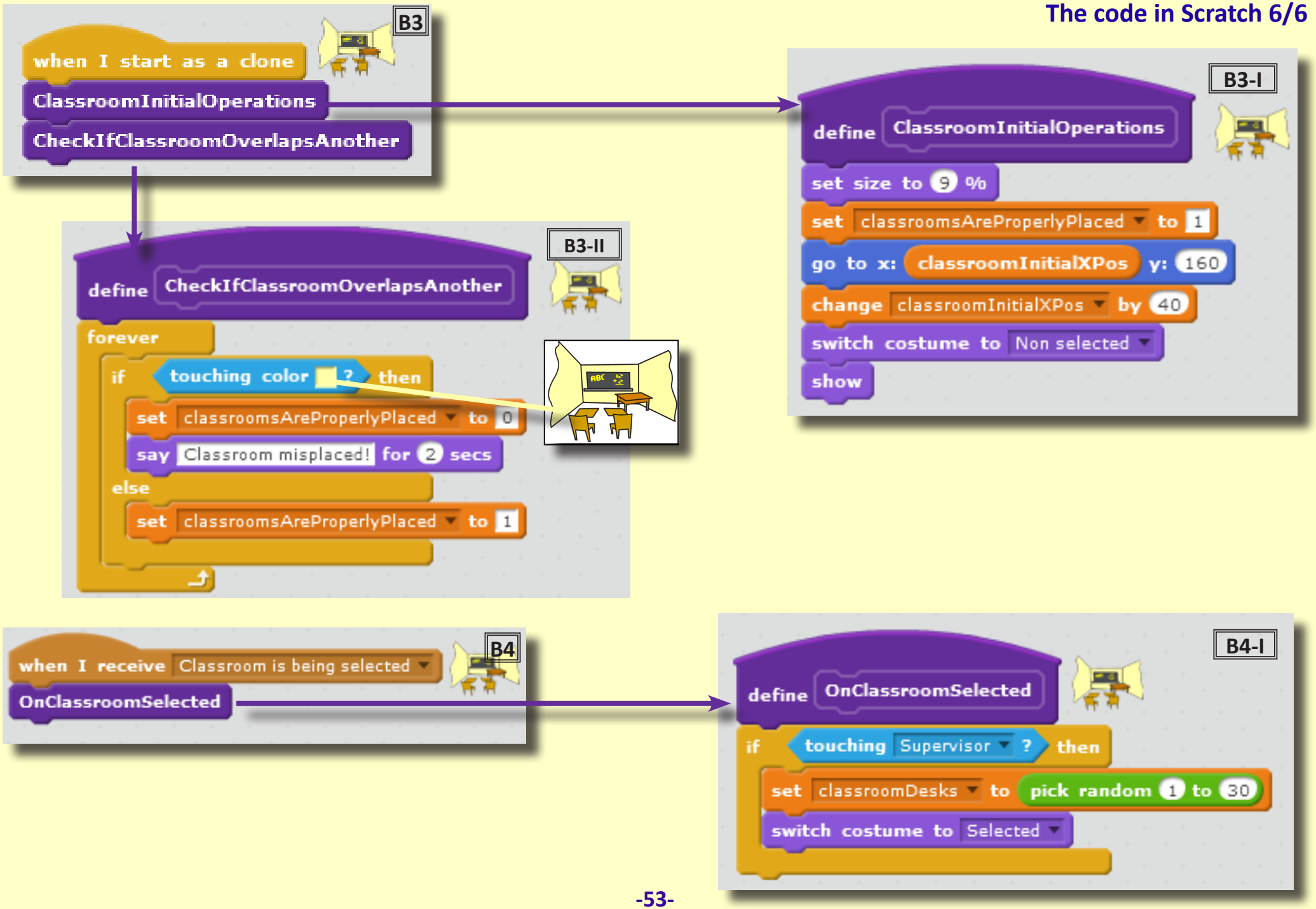

C1
when clicked
InitializeVariables
InitializeSupervisorAppearance
repeat until touching Exit ?
MoveSupervisor
if touching color ? then
CalculateDeskAndSupervisorTotals
FlashSupervisor
ShowFinalMessages
C1-V
define FlashSupervisor
repeat 10
next costume
wait 0.1 secs
Νέα ενδυμασία:
1
ενδυμασία1
103x103
2
ενδυμασία2
103x103
C1-IV
define ShowFinalMessages
say join Total desks in selected classrooms: selectedClassroomsTotalDesks for 4 secs
say join Total supervisors required: totalSupervisors for 4 secs


A1
when clicked
click here
to create classrooms
go to x: -144 y: -119
set numberOfClassrooms to 0
say Click repeatedly to create multiple classrooms. for 4 secs
say Place classrooms along the path. for 4 secs
say Click "Start Supervisor" button for 4 secs

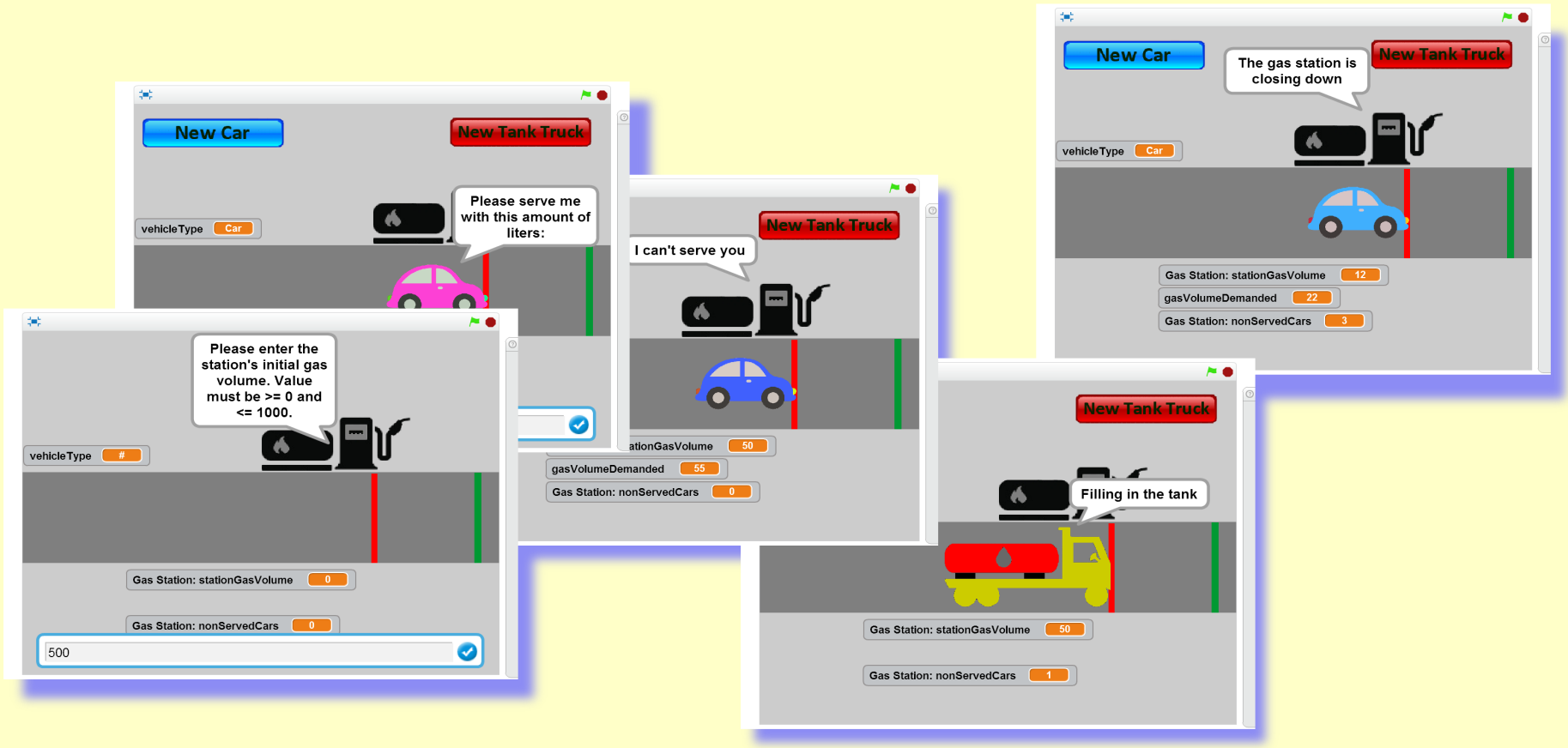

B1
when clicked
go to x: 240 y: -170
hide
set classroomInitialXPos to -210


when this sprite clicked
click here
to create classrooms
if numberOfClassrooms < 19 then
set ghost effect to 50
change numberOfClassrooms by 1
create clone of Classroom
wait 0.3 secs
clear graphic effects
A2

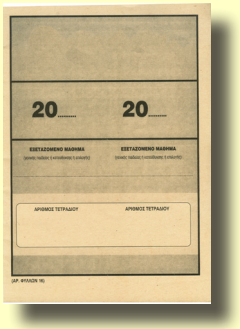

B3
when I start as a clone
ClassroomInitialOperations
CheckIfClassroomOverlapsAnother


B3
when I start as a clone
ClassroomInitialOperations
CheckIfClassroomOverlapsAnother
B3-I
define ClassroomInitialOperations
set size to 9 %
set classroomsAreProperlyPlaced to 1
go to x: classroomInitialXPos y: 160
change classroomInitialXPos by 40
switch costume to Non selected
show
B3-II
define CheckIfClassroomOverlapsAnother
forever
if touching color ? then
set classroomsAreProperlyPlaced to 0
say Classroom misplaced! for 2 secs
else
set classroomsAreProperlyPlaced to 1
B4
when I receive Classroom is being selected
OnClassroomSelected
B4-I
define OnClassroomSelected
if touching Supervisor ? then
set classroomDesks to pick random 1 to 30
switch costume to Selected

4. Gas station operation



Hellenic National Examinations Question



Secondary School Leaving Examination
June 9th 2011
Tested course: Developing applications in a programming environment

Question 3

A gas station stores its gasoline in a 10,000 liter tank. Develop an algorithm which should:

1. Read from the user the gasoline volume (in liters) that the tank initially stores until a valid value is given.

2. For each incoming vehicle read its type ("T" for tank truck supplying the gas station with fuel and "C" for passenger car being refueled in the station with gasoline.)

3. For each incoming vehicle, if it is a tank truck fill the station tank to its maximum capacity.

4. For each incoming vehicle, if it is a passenger car read from the user the amount of gasoline they wish to pump into the car, and if there is enough fuel in the station tank, then the car should be refueled with the requested amount of gasoline, otherwise the car should not be served.

5. The above repetitive process should terminate when either the station tank goes empty or there are three non-served consecutive cars.

6. At the end the algorithm should show:
   a. The average amount of gasoline per served car.
   b. The total amount of gasoline which all tank trucks have refueled the station tank with.

Notes:
- No validation check is required for the vehicle type.
- You may safely assume that at least one car enters the station for which the station tank gasoline is sufficient to serve.

Hellenic National Examinations Question - Tested course: Developing Applications in a Programming Environment - Scratch adaptation

The owner of a gas station requests the development of an automated gasoline management system with the following requirements:

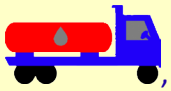
1. When the gas station  begins operating, the application user specifies the original gasoline quantity of the station tank.

2. By clicking one of the two buttons (

, 

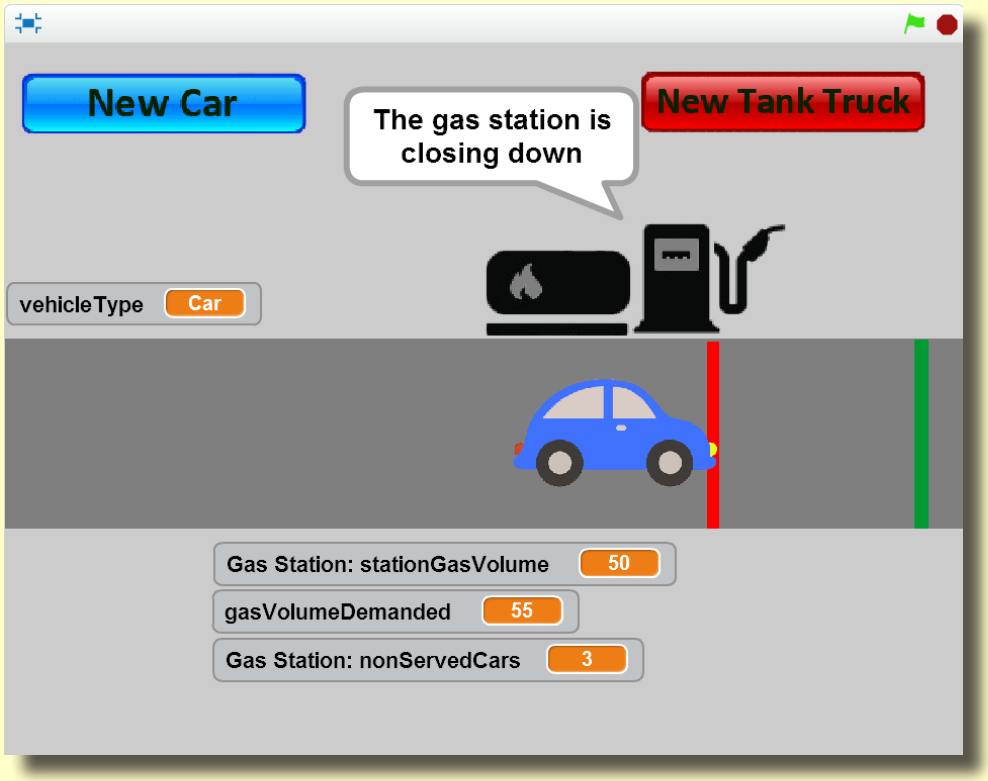
), the user specifies the incoming vehicle type. If the vehicle is:

   a) A Car  , it enters the service lane and when it reaches the pump, it requests a gasoline quantity.
      i) If the station tank gasoline quantity is enough, then the pump fills the car reservoir with the requested quantity and informs the user with an appropriate message.
      ii) If station tank quantity is not enough, the car reservoir is not filled and the user is informed with an appropriate message.

   b) A Tank Truck  , then it enters the service lane and when it reaches the pump it fills up the station tank up to the 1000-liter limit.

   The vehicle subsequently leaves the station, and the service lane becomes available for the next vehicle.

3. The station stops operating when:
   a) Either the station tank quantity becomes 0,
   b) Or three consecutive cars have not been served.



Note: the program must not let a vehicle enter the service lane if it's occupied by another vehicle.

Program Control Flow

1. The Gas Station makes the necessary preparations (initializing its appearance and variables), asks the user to enter the station's initial gasoline volume (a value between 0 and 1000), and then lets all interested parties know that the "Gas station is operating", by broadcasting the corresponding message.

2. This message ("Gas station is operating") is being received by the two buttons "New Car" and "New Tank Truck" which then show.

3. Nothing happens until the user clicks one of the two above buttons.

4. When the "New Car" button is being clicked it checks to see if there isn't another vehicle in the station. If the station is free, then it goes ahead and reserves it so that no other vehicle can use it (by raising the thereIsAVehicleInTheStation flag to 1), resets the pump's gas counter, and finally gives control to a Car clone. Otherwise, it informs the user that "*There is another vehicle in the gas station*". Upon Car clone creation:

a. The clone gets its "personalized" color, sets its vehicle type to "Car" and starts moving toward the gas station. When it reaches it, it asks the user for the amount of gasoline they want.

b. It then broadcasts to all interested parties the message "Transact with the gas station" and waits for all actions triggered by the reception of that message to complete.

c. That message is subsequently being received by the Gas Station, which, taking control of the program flow, checks the vehicle type.

d. If it's a Car, the Gas Station scenario proceeds with a car transaction, during which if the gasoline in the station's tank is enough to satisfy the user request, then that amount is being subtracted from the station tank, and the number of cars that haven't been served in a row is reset (to 0). Otherwise, the user is being informed they "*I can't be served*" and the counter of non-served cars is incremented by one.

e. Once the Gas Station scenario handling the message "Transact with the gas station" is done (after returning from the CheckGasStationOperation procedure), program control returns back to the Car clone scenario and the DepartFromGasStation procedure is called to move the Car to the right and release the station (by lowering the thereIsAVehicleInTheStation flag to 0) when the Car passes the green line. Finally, the scenario deletes the Car clone, making the car disappear from the station.

Program Control Flow

5. When the "New Tank Truck" button is being clicked it, too, checks to see if there isn't another vehicle in the station. If again the station is free then it goes ahead and reserves it, and then creates a Tank Truck clone. Otherwise, it informs the user that "*There is another vehicle in the gas station***". Upon Tank Truck clone creation:**

a. The clone sets its vehicle type to "Tank Truck", starts moving toward the gas station and when it reaches it, lets the user know that "*it will start filling the station tank***".**

b. It then broadcasts to all interested parties the message "Transact with the gas station" and waits (for all actions triggered by the reception of that message to complete).

c. That message ("Transact with the gas station") is subsequently being received by the Gas Station, which, upon taking control of the program flow, checks for the incoming vehicle type.

d. This time the vehicle is a Tank Truck, so the Gas Station scenario proceeds with a tank truck transaction to fill the station tank up to the 1000-liter limit.

e. Once the tank truck transaction is completed in the Gas Station scenario, program control returns back to the Tank Truck clone scenario and the DepartFromGasStation procedure is called the same way it's called by the Car (see step 4e above).

6. Finally, the Gas Station, in the case of the arriving vehicle being a Car, checks for its future operation so that, if the station's tank is empty or there are three consecutive cars that have not been served, it broadcasts the message "Gas station is closing down", thus letting the two sprite buttons "New Car" and "New Tank Truck" know about it, and then waits for them to finish handling the message.

7. When the two buttons ("New Car" and "New Tank Truck") receive the message "Gas station is closing down" they just hide.

Sprites' Functional Analysis

Gas Station

When the green flag is being clicked:

- Takes its position on the screen and then hides.
- Makes the necessary preparations (sets its appearance, initializes its variables),

asks the user to specify the initial gas volume of its tank (a value from 0 to 1000 liters), and finally broadcasts to all parties interested the message "Gas station is operating".

When it receives the message "Transact with the gas station",
checks whether the incoming vehicle is a car or a tank truck.

- Upon determining that the incoming vehicle is a car, it proceeds to transact with it: if the quantity of the gas in the station tank is enough to serve the customer,
  then the sprite removes that quantity from the station tank;
  otherwise it informs the user that it cannot serve them and increments by one the non-served-cars counter.

- If the incoming vehicle is a tank truck, it goes ahead and transacts with it, meaning that it fills the station tank up to the 1000-liter limit.

- After the transaction has completed either way, the station sprite, in the case of the vehicle being a car, checks its future operation: if either its tank is empty or three customers were not served in a row, then it informs the user that "*The gas station is closing down*", and lets the "New Car" and "New Tank Truck" buttons know about it by broadcasting the message "Gas station is closing down" and waiting for those two buttons to finish handling the message before it continues.

Sprites' Functional Analysis

"New Car" button

When the green flag is clicked

- **it hides.**

When it receives the message "Gas station is operating"

- **itshows.**

When it is clicked,

- **it checks if there is not another vehicle in the station. If the station is free,**

  then

  - **it goes ahead and reserves it so that no other vehicle can use it (by raising the thereIsAVehicleInTheStation flag to 1),**
  - **resets the pump's gas counter, and finally,**
  - **gives control to the Car (or rather to a Car clone),**

  otherwise,

  - **it informs the user that "*There is another vehicle in the gas station*".**

When it receives the message "Gas station is closing down"

- **it just hides itself.**

"NewTank Truck" button

When the green flag is clicked

- **it hides.**

When it receives the message "Gas station is operating"

- **it shows.**

When it is clicked,

- **it checks if there is not another vehicle in the station. If the station is free,**

  then

  - **it goes ahead and reserves it so that no other vehicle can use it,**
  - **hides the pump's gas counter (since that is not applicable in the case), and then gives control to a Tank Truck clone,**

  otherwise,

  - **it informs the user that "*There is another vehicle in the gas station*".**

When it receives the message "Gas station is closing down"

- **it hides too.**

Sprites' Functional Analysis

Car

When the green flag is clicked:

- It takes its position on screen and hides.

When a Car clone is being created:

- It gets its "personalized" color, sets its vehicle type to "Car" and starts moving to the gas station until it reaches it. It then asks the user for the amount of gasoline they want.
- It subsequently broadcasts to all interested parties the message "Transact with the gas station" and waits for all actions triggered by the reception of that message to finish.
- When all gas station transactions are completed, the Car leaves the station. Specifically, the departing procedure (DepartFromGasStation) moves the Car to the right, releasing the station (by lowering the thereIsAVehicleInTheStation flag to 0) when the Car goes past the green line. Finally, the scenario deletes the current Car clone, making thus the car disappear from the station.

Tank Truck

When the green flag is clicked:

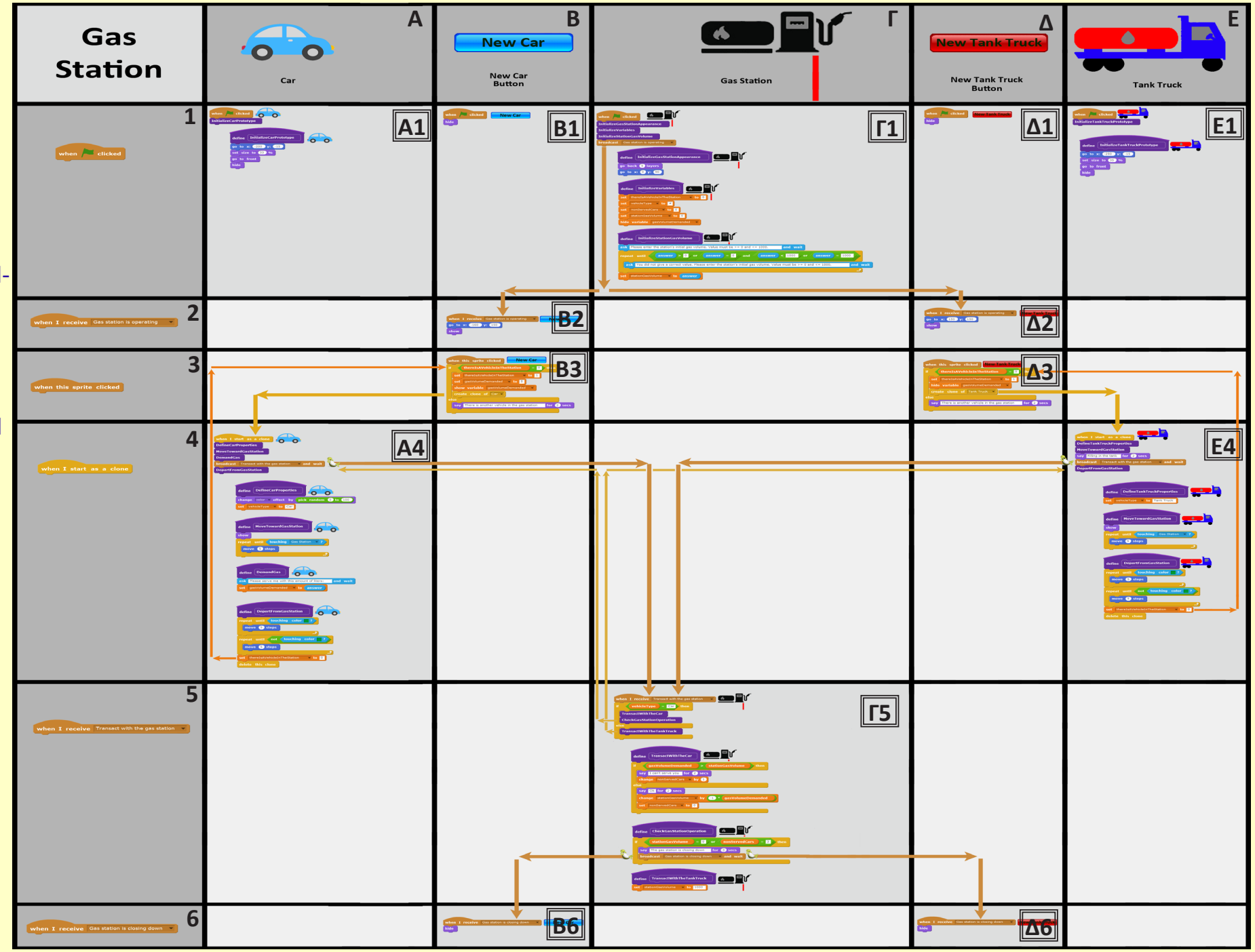
- It takes its position on screen and hides.

When a Tank Truck clone is being created:

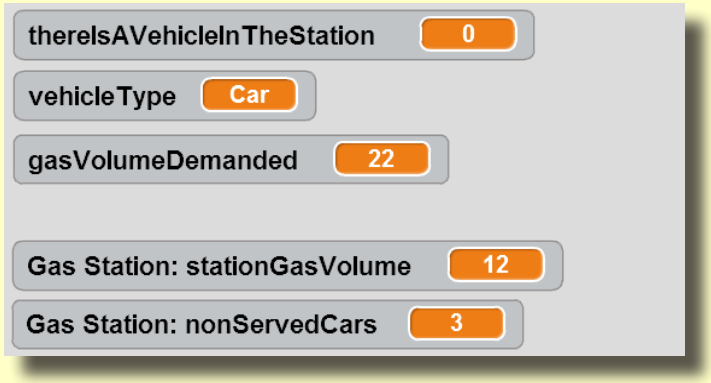
- It sets its vehicle type to "Tank Truck" and then begins moving to the gas station until it reaches it, upon which it informs it that it is filling in the tank.
- It subsequently broadcasts to all interested parties the message "Transact with the gas station" and waits for all actions triggered by the reception of that message to finish.
- When all gas station transactions are completed, the Tank Truck leaves the station. Specifically, the departing procedure (DepartFromGasStation) moves the Tank Truck to the right, releasing the station (by lowering the thereIsAVehicleInTheStation flag to 0) when the truck goes past the green line. Finally, the scenario deletes the Tank Truck clone, making thus the truck disappear from the station.

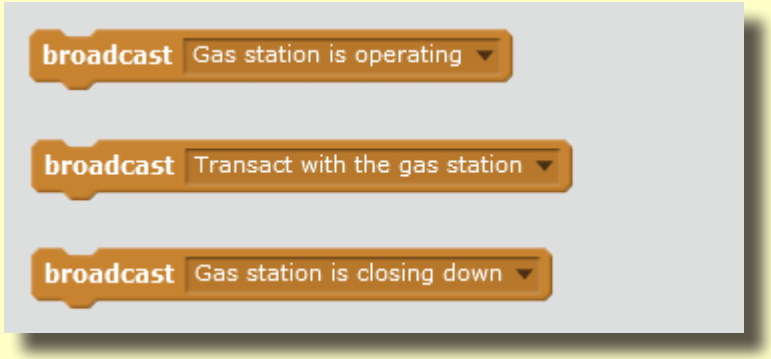
CodOrama

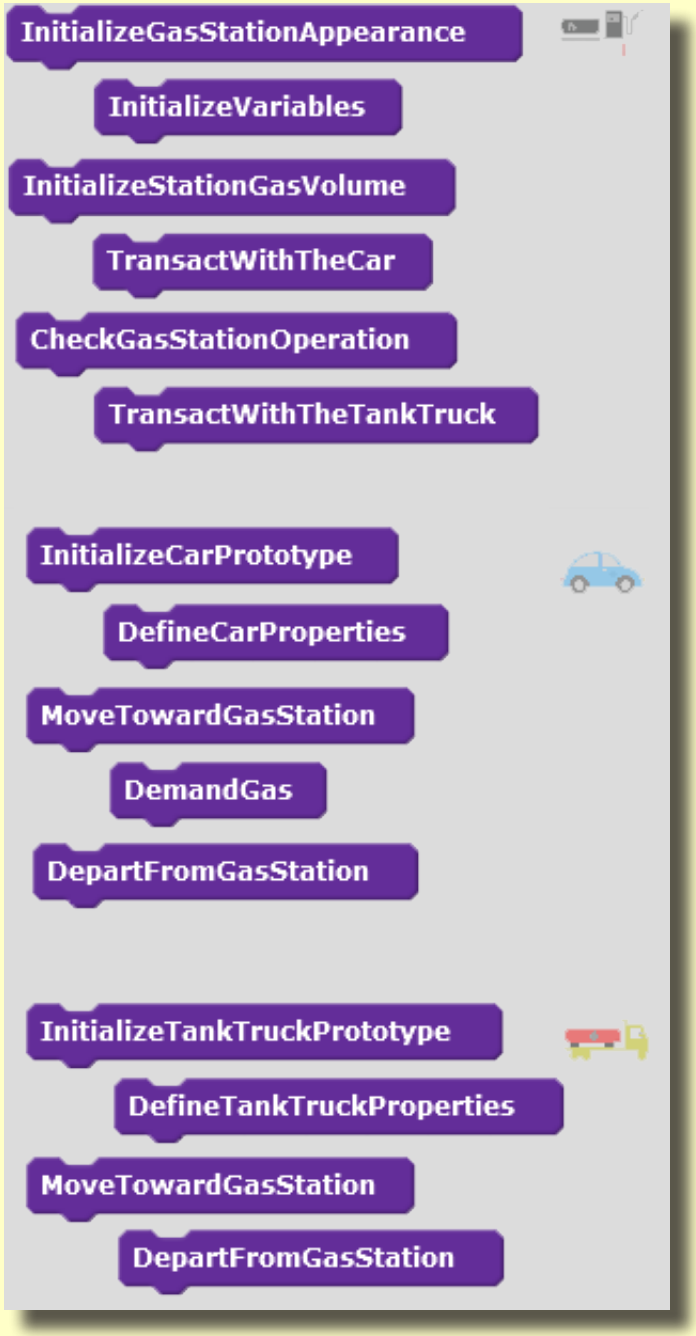
The table on this page, where we can see the various code modules and their interconnections, may be used in order to understand these scenarios.



Variables - messages - procedures



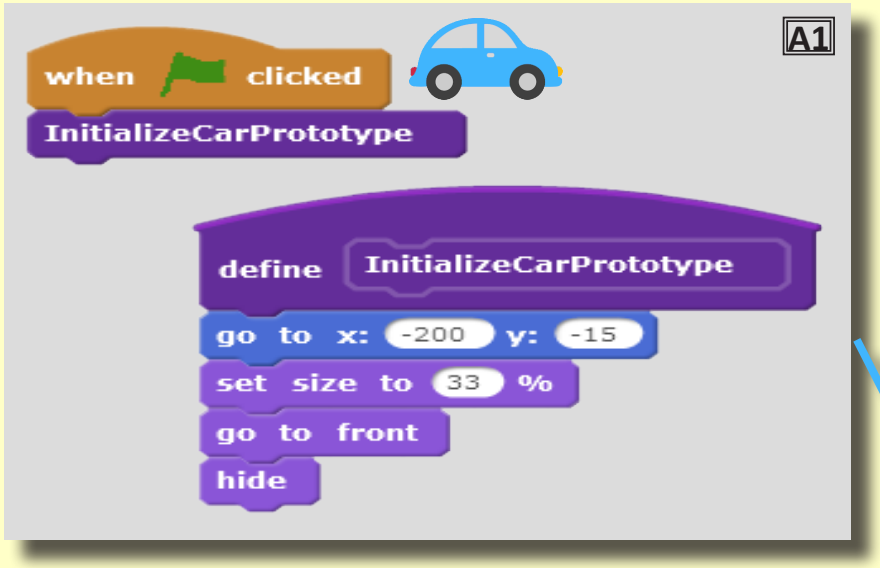


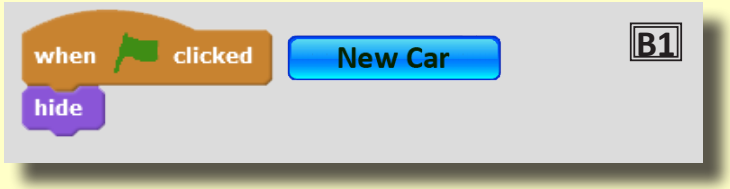


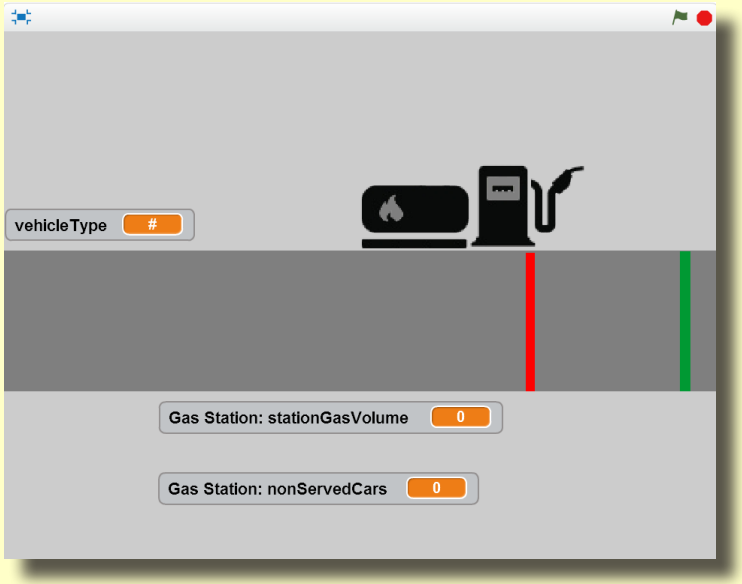
The code in Scratch

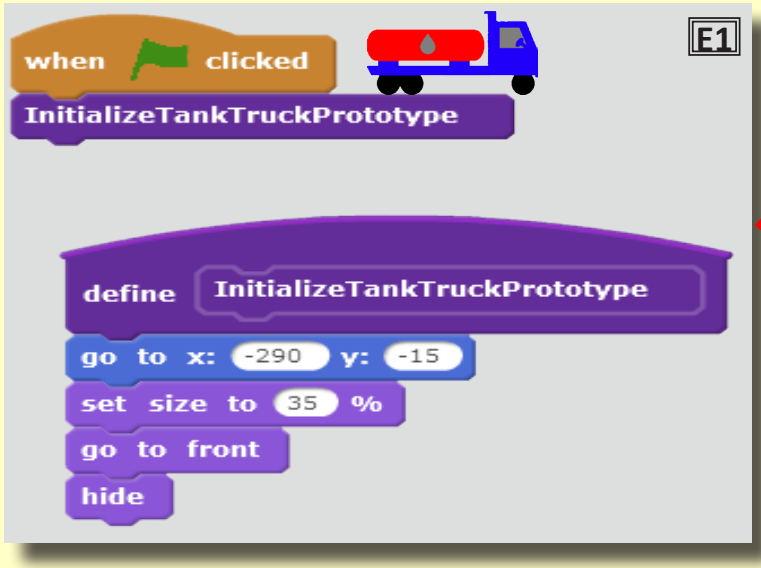
CodOrama: https://prezi.com/0wtdhgw9bs1z/gas-station/ or http://bit.ly/1X7tM2k

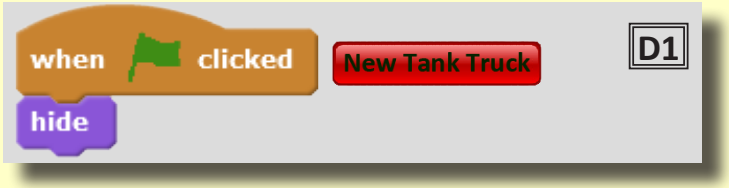
Code: https://scratch.mit.edu/projects/82682436/ or http://bit.ly/1Qy67nX

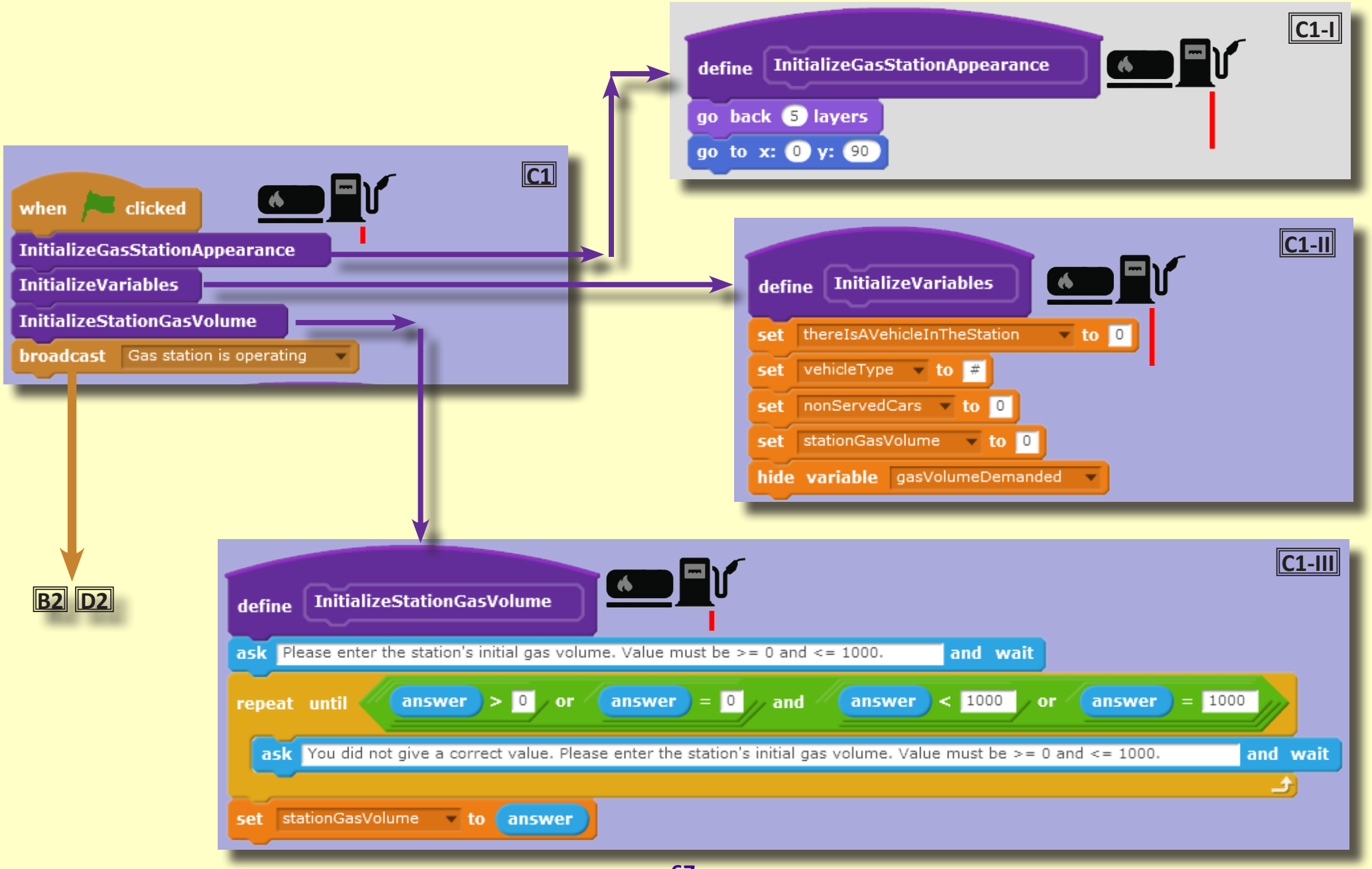

A1
when clicked
InitializeCarPrototype
define InitializeCarPrototype
go to x: -200 y: -15
set size to 33 %
go to front
hide

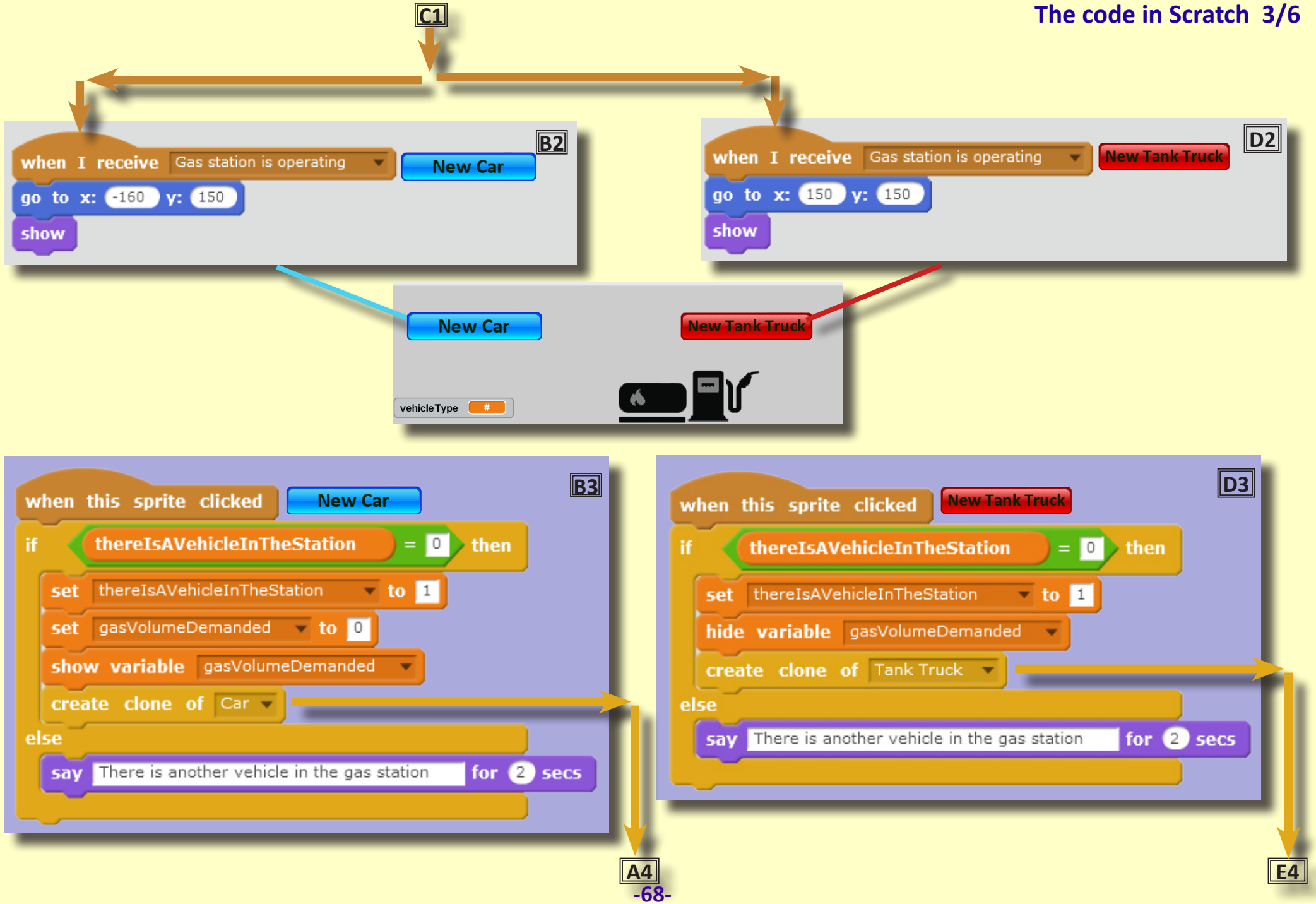

B1
when clicked
New Car
hide

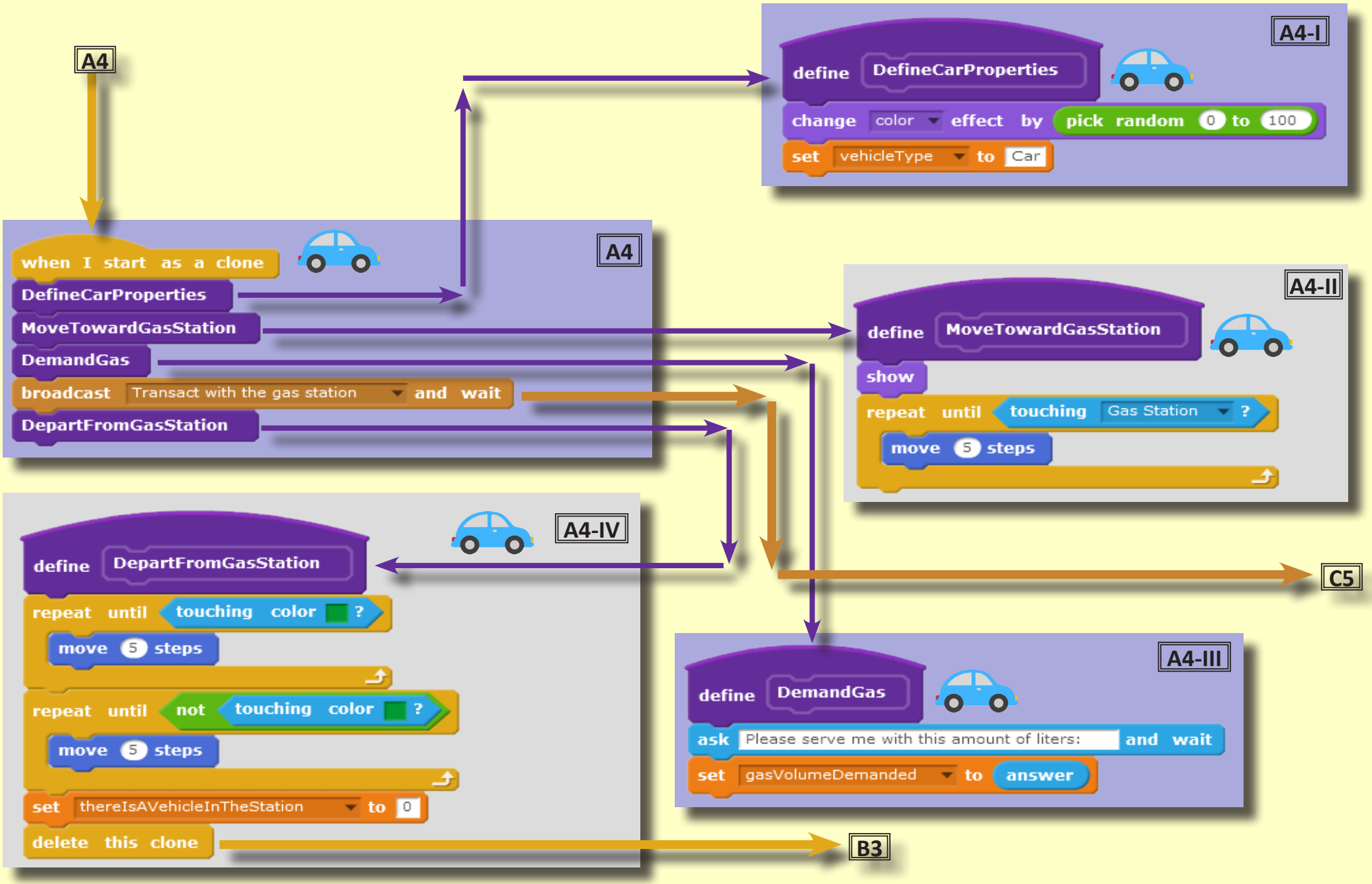

vehicleType #
Gas Station: stationGasVolume 0
Gas Station: nonServedCars 0


E1
when clicked
InitializeTankTruckPrototype
define InitializeTankTruckPrototype
go to x: -290 y: -15
set size to 35 %
go to front
hide


D1
when clicked
New Tank Truck
hide


C1
when clicked
InitializeGasStationAppearance
InitializeVariables
InitializeStationGasVolume
broadcast Gas station is operating
B2
D2
C1-I
define InitializeGasStationAppearance
go back 5 layers
go to x: 0 y: 90
C1-II
define InitializeVariables
set thereIsAVehicleInTheStation to 0
set vehicleType to #
set nonServedCars to 0
set stationGasVolume to 0
hide variable gasVolumeDemanded
C1-III
define InitializeStationGasVolume
ask Please enter the station's initial gas volume. Value must be >= 0 and <= 1000. and wait
repeat until answer > 0 or answer = 0 and answer < 1000 or answer = 1000
ask You did not give a correct value. Please enter the station's initial gas volume. Value must be >= 0 and <= 1000. and wait
set stationGasVolume to answer


C1
B2
when I receive Gas station is operating
New Car
go to x: -160 y: 150
show
D2
when I receive Gas station is operating
New Tank Truck
go to x: 150 y: 150
show
New Car
New Tank Truck
vehicleType #
B3
when this sprite clicked
New Car
if thereIsAVehicleInTheStation = 0 then
set thereIsAVehicleInTheStation to 1
set gasVolumeDemanded to 0
show variable gasVolumeDemanded
create clone of Car
else
say There is another vehicle in the gas station for 2 secs
A4
D3
when this sprite clicked
New Tank Truck
if thereIsAVehicleInTheStation = 0 then
set thereIsAVehicleInTheStation to 1
hide variable gasVolumeDemanded
create clone of Tank Truck
else
say There is another vehicle in the gas station for 2 secs
E4


A4
A4-I
define DefineCarProperties
change color effect by pick random 0 to 100
set vehicleType to Car
A4
when I start as a clone
DefineCarProperties
MoveTowardGasStation
DemandGas
broadcast Transact with the gas station and wait
DepartFromGasStation
A4-II
define MoveTowardGasStation
show
repeat until touching Gas Station ?
move 5 steps
A4-IV
define DepartFromGasStation
repeat until touching color ?
move 5 steps
repeat until not touching color ?
move 5 steps
set thereIsAVehicleInTheStation to 0
delete this clone
C5
A4-III
define DemandGas
ask Please serve me with this amount of liters: and wait
set gasVolumeDemanded to answer
B3

E4

E4

when I start as a clone

DefineTankTruckProperties

MoveTowardGasStation

say Filling in the tank for 2 secs

broadcast Transact with the gas station and wait

DepartFromGasStation

E4-I

define DefineTankTruckProperties

set vehicleType to Tank Truck

E4-II

define MoveTowardGasStation

show

repeat until touching Gas Station ?

move 5 steps

C5

E4-III

define DepartFromGasStation

repeat until touching color ?

move 5 steps

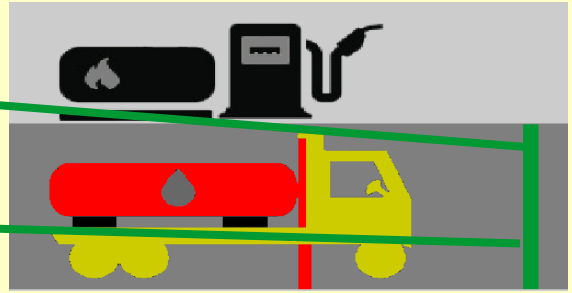
repeat until not touching color ?

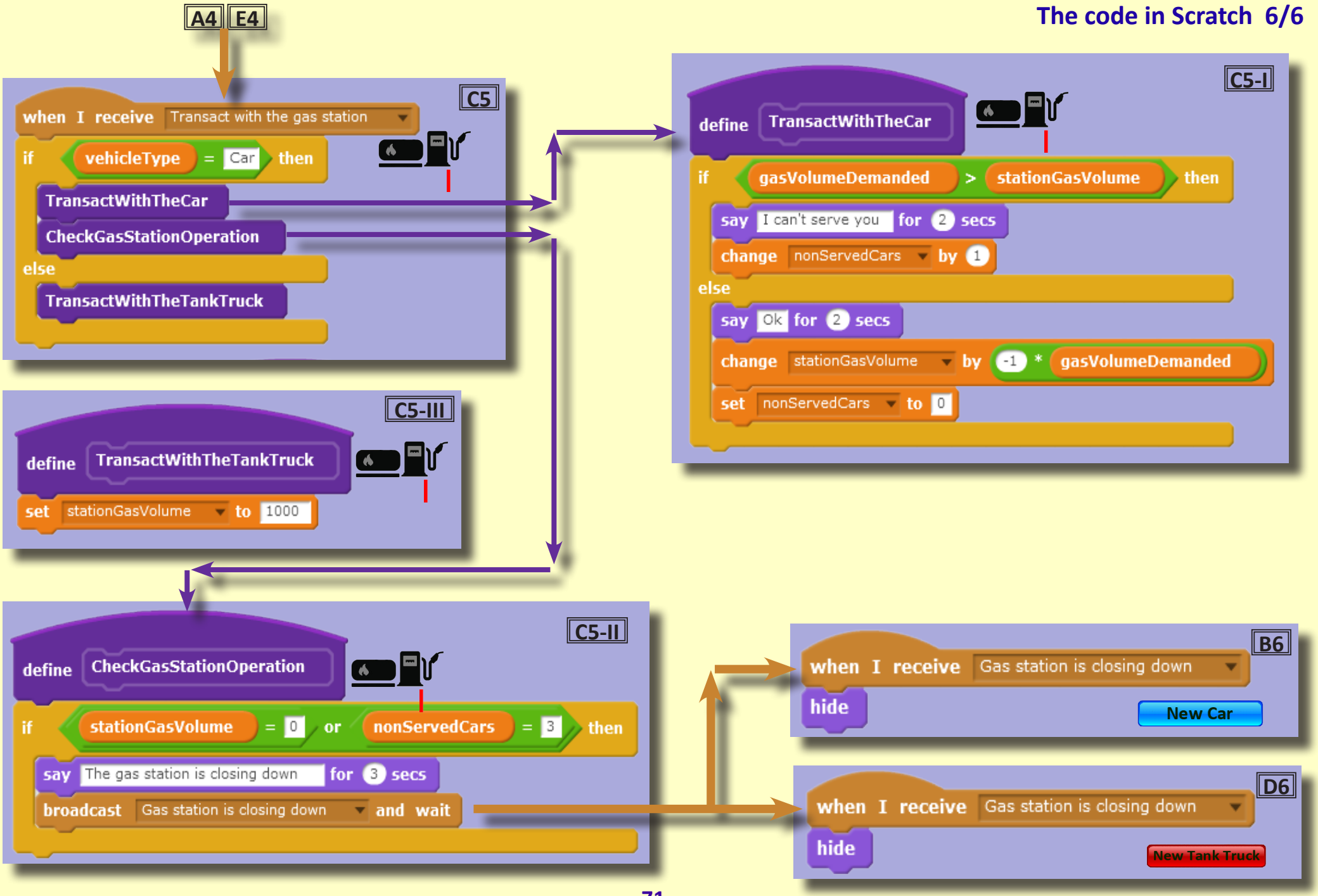
move 5 steps

set thereIsAVehicleInTheStation to 0

delete this clone

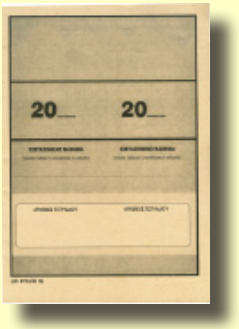
D3




A4
E4
C5
when I receive Transact with the gas station
if vehicleType = Car then
TransactWithTheCar
CheckGasStationOperation
else
TransactWithTheTankTruck
C5-I
define TransactWithTheCar
if gasVolumeDemanded > stationGasVolume then
say I can't serve you for 2 secs
change nonServedCars by 1
else
say Ok for 2 secs
change stationGasVolume by -1 * gasVolumeDemanded
set nonServedCars to 0
C5-III
define TransactWithTheTankTruck
set stationGasVolume to 1000
C5-II
define CheckGasStationOperation
if stationGasVolume = 0 or nonServedCars = 3 then
say The gas station is closing down for 3 secs
broadcast Gas station is closing down and wait
B6
when I receive Gas station is closing down
hide
New Car
D6
when I receive Gas station is closing down
hide
New Tank Truck

5. The Game: Rock-Paper-Scissors

Hellenic National Examinations Question



Secondary School Leaving Examination
July4th 2007
Tested course: Developing applications in a programming environment

Question 3

The classic "Rock-Paper-Scissors game is played by two players. In each round of the game, each player selects one of ROCK, PAPER, SCISSORSand presents his/her selection at the same time with the other player. In the case of the two players having made the same selection the round ends with a tie. The game continues with repeated rounds until one of the players quits. The winner is the player with the most wins. If the two players have the same number of wins, the game ends with a tie.

Write an algorithm that will read the names of the two players and implement the above game as follows:

A. In each round:

1. It will read each player's selection out of ROCK, PAPER, SCISSORS or END. (No validation check for input values is required.)
2. It will compare the players' selections and will then announce the winner of the game or declare a tie.

B. It will end the game when at least one of the players chooses to respond with END in a round.

Γ. It will show the name of the winner – if there is one, or, if there is no winner, display the message "Game ends with a tie".

Hellenic National Examinations Question - Tested course: Developing Applications in a Programming Environment - Scratch adaptation


Nick and Maria will be playing the well-known Rock-Paper-Scissors game. The game starts by clicking the Start game button **Start Game**.

Then:

- The two player sprites make their selection (Rock, Scissors, or Paper).

- The Computer sprite waits for their responses and then announces the winner of the current round (based on the rule: Rock wins over Scissors, Scissors win over Paper and Paper wins over Rock), or if the players make the same selection declares a tie for this round.

- Then the End Game button **End Game** appears for a few seconds to let the user click it in order to end the game. If the button is not clicked, it hides and the above process is repeated.

- When the user decides to end the game (by clicking the End Game button), the Computer sprite declares the winner of the game or a tie message, if there's no winner.

During the game the following information is displayed: Nick's wins, Maria's wins, and the number of rounds that have been played so far. Also, the Start Game button must hide after being initially clicked since the game has already begun and there is no point in using it again.

Program Control Flow

After the Computer sprite has made the necessary preparations (initializing variables) when the program starts, it broadcasts the message "Activate Start Button", relinquishing program control to the Start Button. When the latter receives the message it runs the required commands to show up in the appropriate position.

Nothing transpires until the user clicks the Start Button.

When the Start Button clicked, all interested sprites receive the message "Let the round begin", while the button itself becomes hidden.

The message "Let the round begin" is received by the following sprites:

- Nick: upon receiving the message "Let the round begin", it randomly selects a value in the range 1-3 (1 corresponding to Rock, 2 to Scissors and 3 to Paper), and then displays its selection in the form of the corresponding word and informs that it made its selection.
- Maria: upon receiving the message "Let the round begin", it randomly selects a value in the range 1-3 (1 corresponding to Rock, 2 to Scissors and 3 to Paper), and then displays its selection in the form of the corresponding word and informs that it made its selection.
- The Computer sprite – upon receiving the message "Let the round begin":
    - it waits for both the players to make their selection.
    - Having subsequently had both players' selections, it determines either the winner of the round or that there is a tie.
    - It then releases program control to the "End Game" button by broadcasting the message "Show End Game Button" and waiting for the "End Game" scenario handling that message to return. Finally, it broadcasts the message "Let the round begin" to all sprites, in order to initiate the next round.

The End Game Button shows up for 4 seconds, during which time the user may click it, and then hides if it's not clicked. In that case, program control goes back to the Computer sprite.

If the End Game Button is clicked it sends the "Announce winner" message, which is received by the Computer sprite that gains control of the program, while the "End Game Button" sits around waiting to get the control back. After getting back the control of the program flow, the button stops all scenarios in the program.

When the Computer sprite receives the message "Announce winner" it finds and announces the winner of the game.

Sprites' Functional Analysis

Computer

When the user clicks the green flag

- TheComputer sprite makes the necessary preparations (initializing variables) in order to start the game.
- It then broadcasts the message "Activate Start Button", relinquishing program control to the Start Button.

When it receives the message "Let the round begin"

- It waits for both players to make their selection.
- Having subsequently had both players' selections, it determines whether there is a tie or the winner of this round
- It then gives program control to the "End Game" button by broadcasting the message "Show End Game Button" and waiting for the "End Game" scenario handling that message to return.
- Finally, it broadcasts the message "Let the round begin" to all sprites in order to initiate the next round.

When it receives the message "Announce winner"

- it determines and announces the winner of the game.

Start Button

When it receives the message "Activate Start Button"

- it executes all necessary actions to show itself.

When it is clicked

- the user message "*The game is starting. The players will be making their selections*" is displayed,
- then the message "Let the round begin" is broadcast
- and the button subsequently hides.

Nick

When it receives the message "Let the round begin"

- it selects a random number in the range 1-3 corresponding to the values Rock, Scissors, Paper, respectively,
- then shows its selection and
- informs that it made its selection.

Maria

When it receives the message "Let the round begin",

- it selects a random number from the range 1-3 corresponding to the values Rock, Scissors, Paper, respectively,
- shows its selection accordingly and
- informs that it made its selection.

End Game Button

When it receives the "Show End Game Button" message

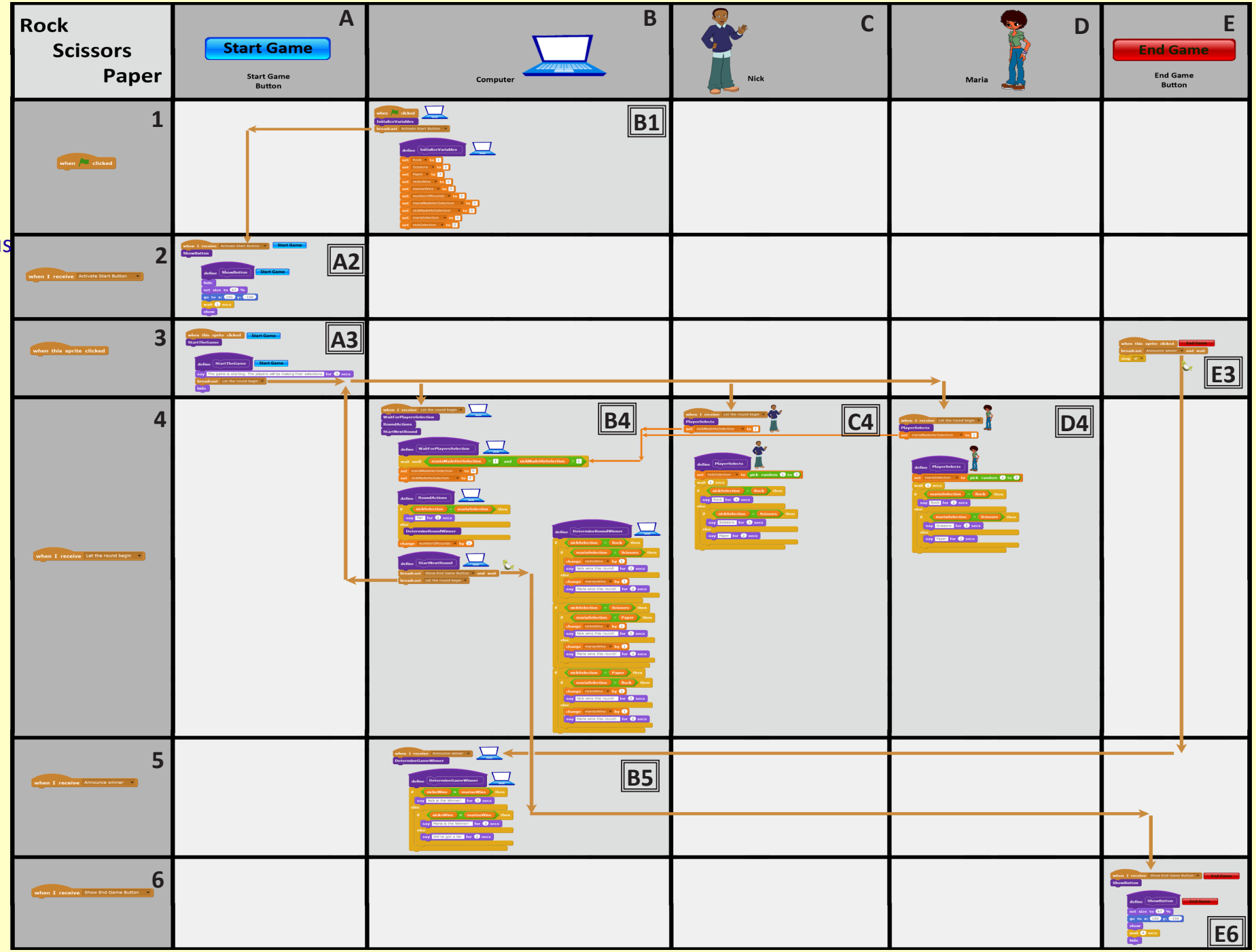
- it shows up for 4 seconds, during which time the user can click it, and then hides.
- If it's not clicked, program control goes back to the Computer sprite.

If it's clicked then

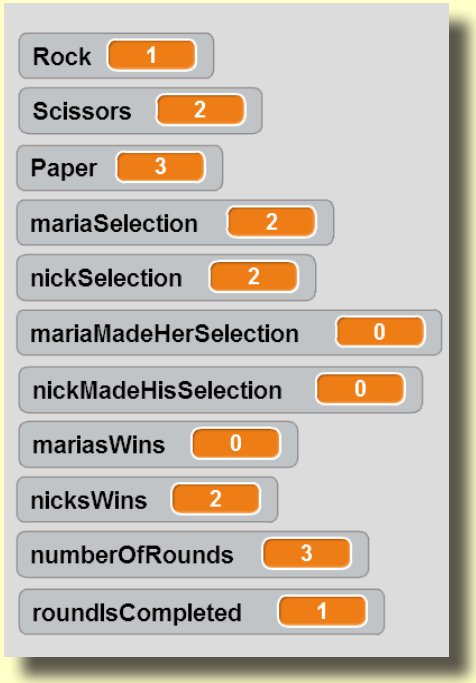
- It sends the "Announce winner" message, which is received by the Computer sprite that gains control of the program,
- while the "End Game Button" sits around waiting to get back the control.
- After getting back the control of the program flow, the button stops all scenarios in the program.

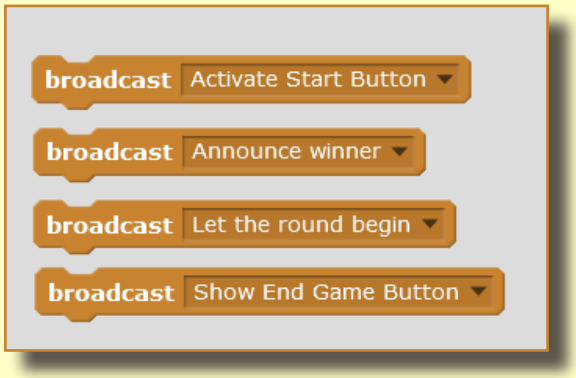
CodOrama

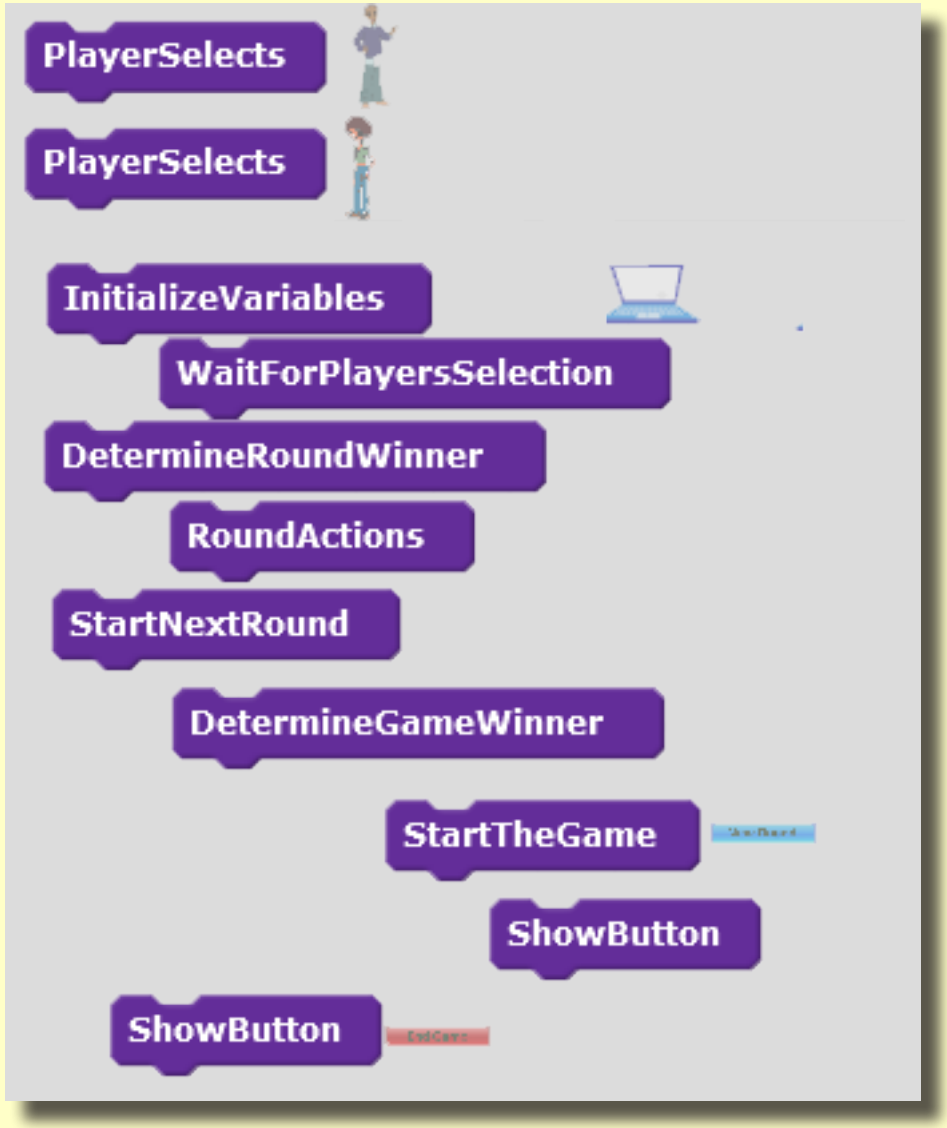
The table on this page, where we can see the various code modules and their interconnections, may be used in order to understand these scenarios.



Variables - messages - procedures



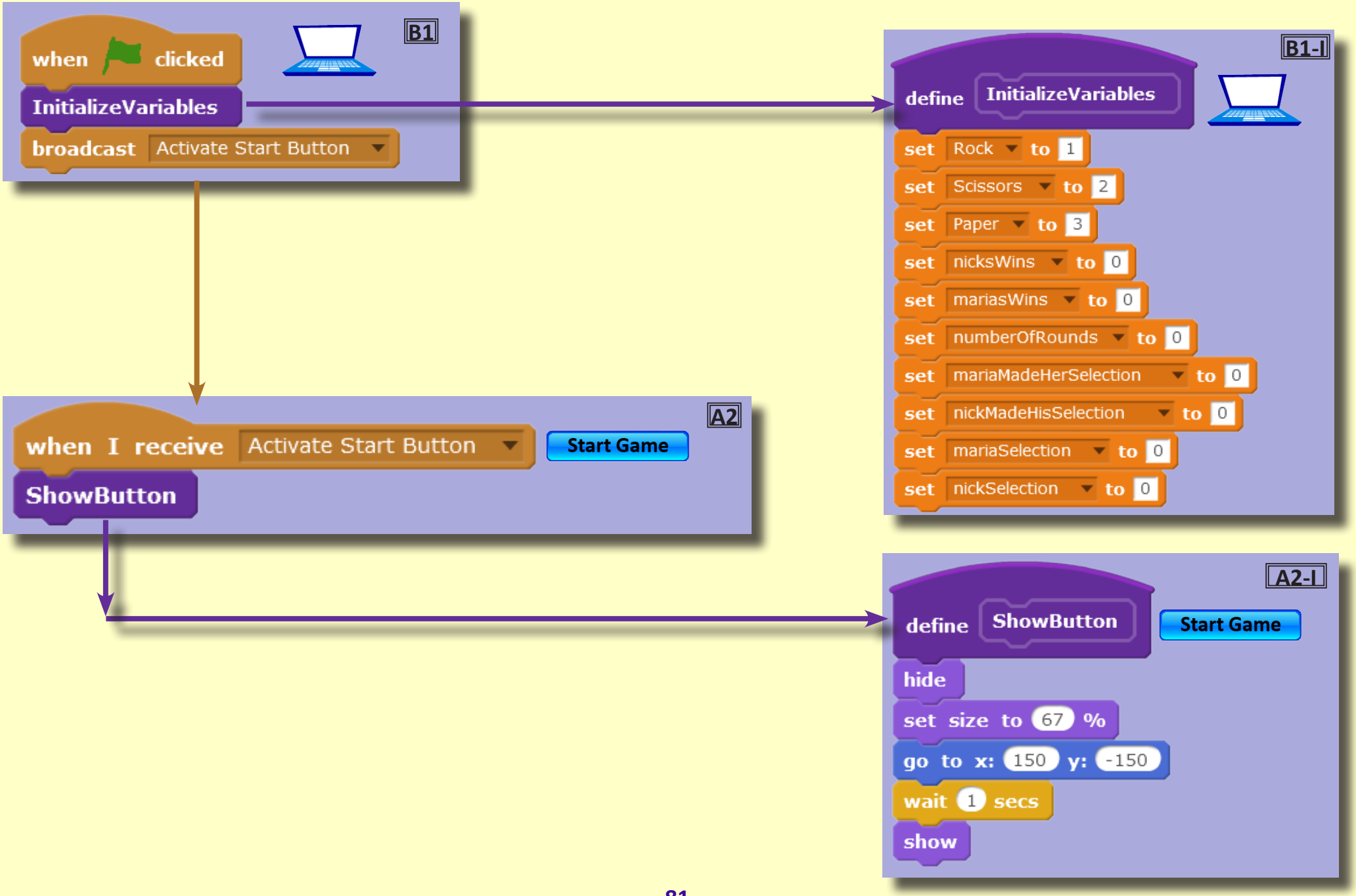


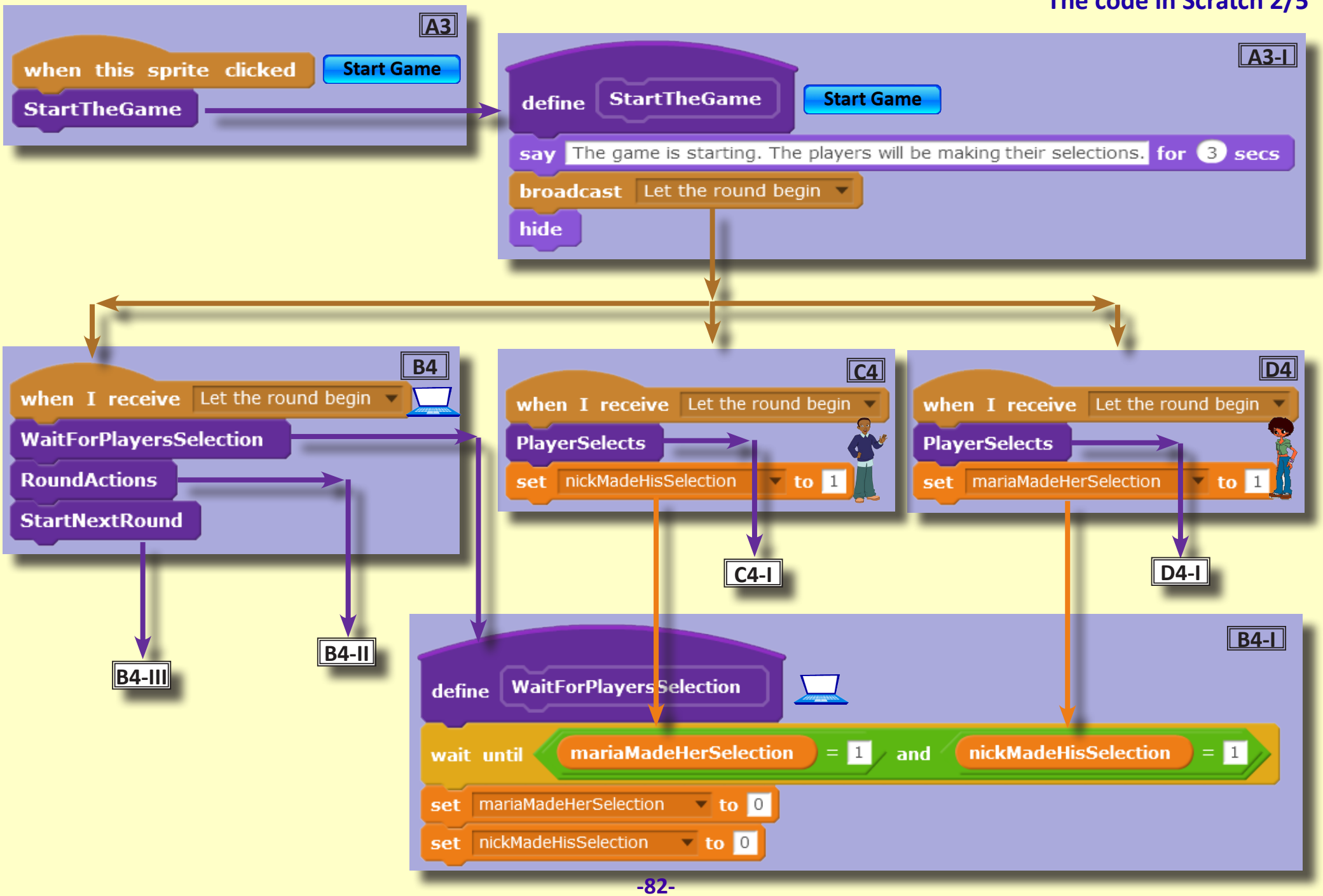


The code in Scratch

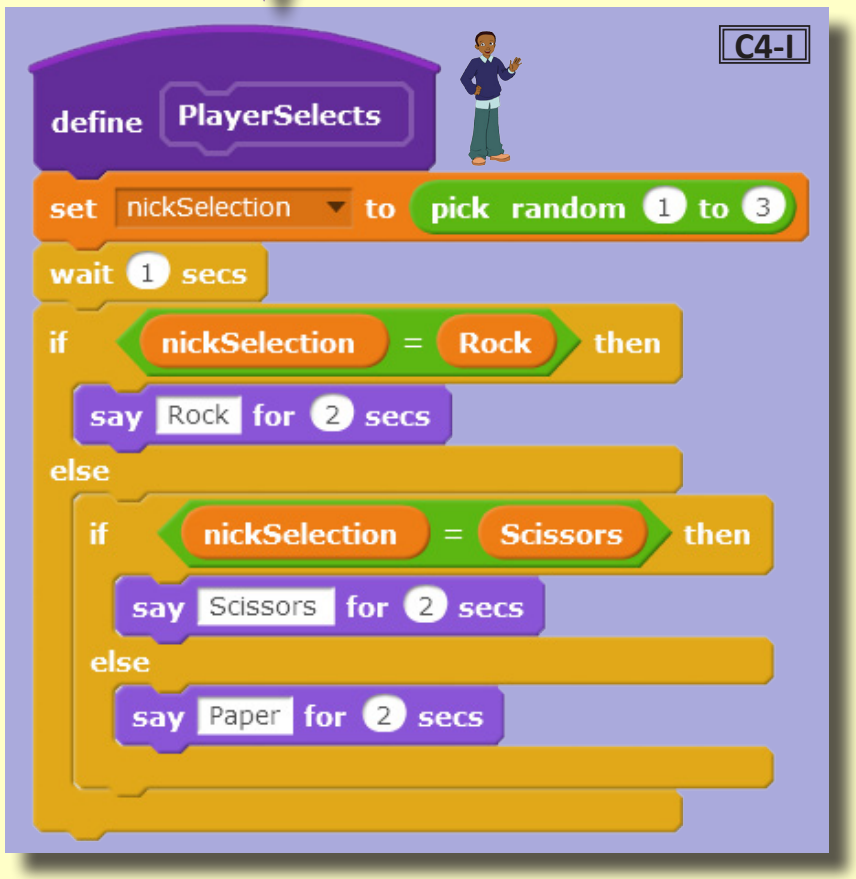
CodOrama: https://prezi.com/ojyrztdqkgtk/rock-scissors-paper/ or http://bit.ly/1NK9gTQ

Code: https://scratch.mit.edu/projects/82684622/ or http://bit.ly/1NeSvx6

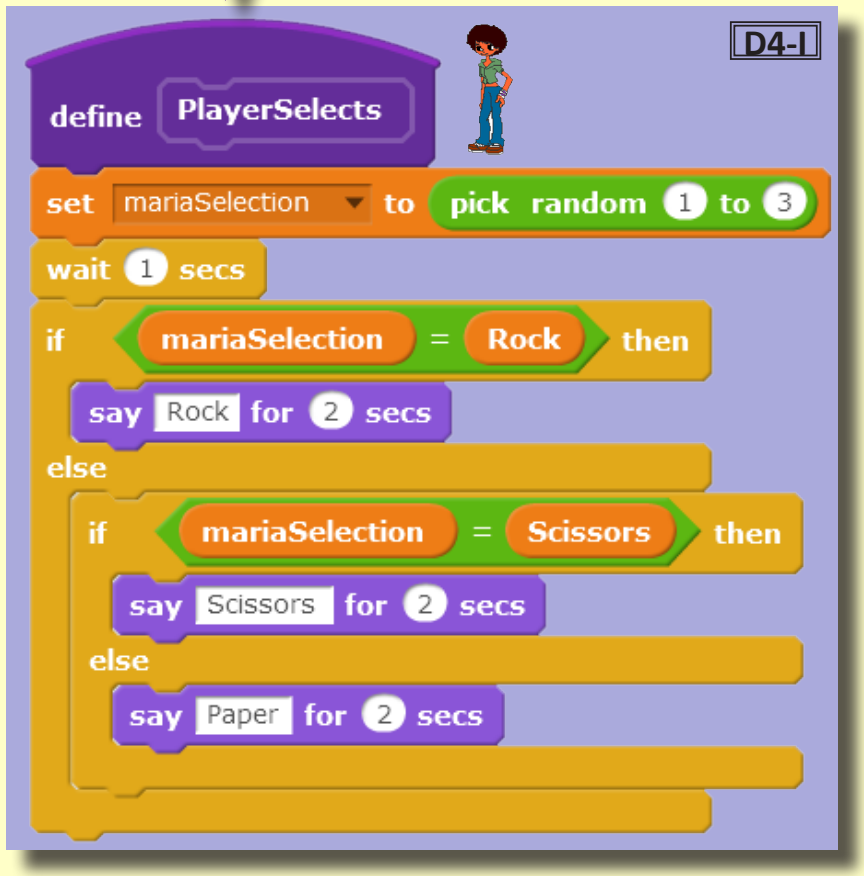

B1
when clicked
InitializeVariables
broadcast Activate Start Button
B1-I
define InitializeVariables
set Rock to 1
set Scissors to 2
set Paper to 3
set nicksWins to 0
set mariasWins to 0
set numberOfRounds to 0
set mariaMadeHerSelection to 0
set nickMadeHisSelection to 0
set mariaSelection to 0
set nickSelection to 0
A2
when I receive Activate Start Button
Start Game
ShowButton
A2-I
define ShowButton
Start Game
hide
set size to 67 %
go to x: 150 y: -150
wait 1 secs
show

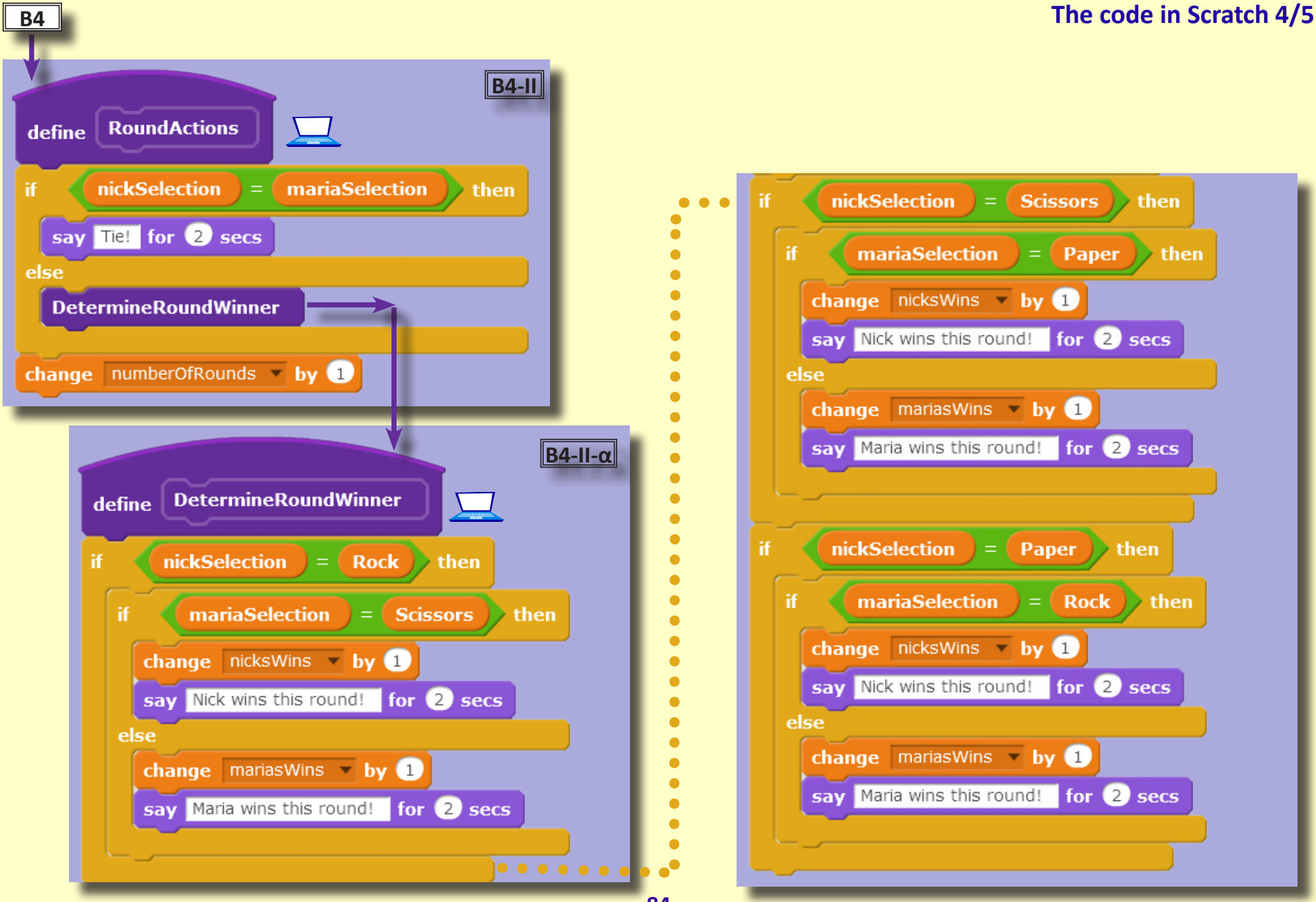

A3
when this sprite clicked
Start Game
StartTheGame
A3-I
define StartTheGame
Start Game
say The game is starting. The players will be making their selections. for 3 secs
broadcast Let the round begin
hide
B4
when I receive Let the round begin
WaitForPlayersSelection
RoundActions
StartNextRound
B4-III
B4-II
C4
when I receive Let the round begin
PlayerSelects
set nickMadeHisSelection to 1
C4-I
D4
when I receive Let the round begin
PlayerSelects
set mariaMadeHerSelection to 1
D4-I
B4-I
define WaitForPlayersSelection
wait until mariaMadeHerSelection = 1 and nickMadeHisSelection = 1
set mariaMadeHerSelection to 0
set nickMadeHisSelection to 0

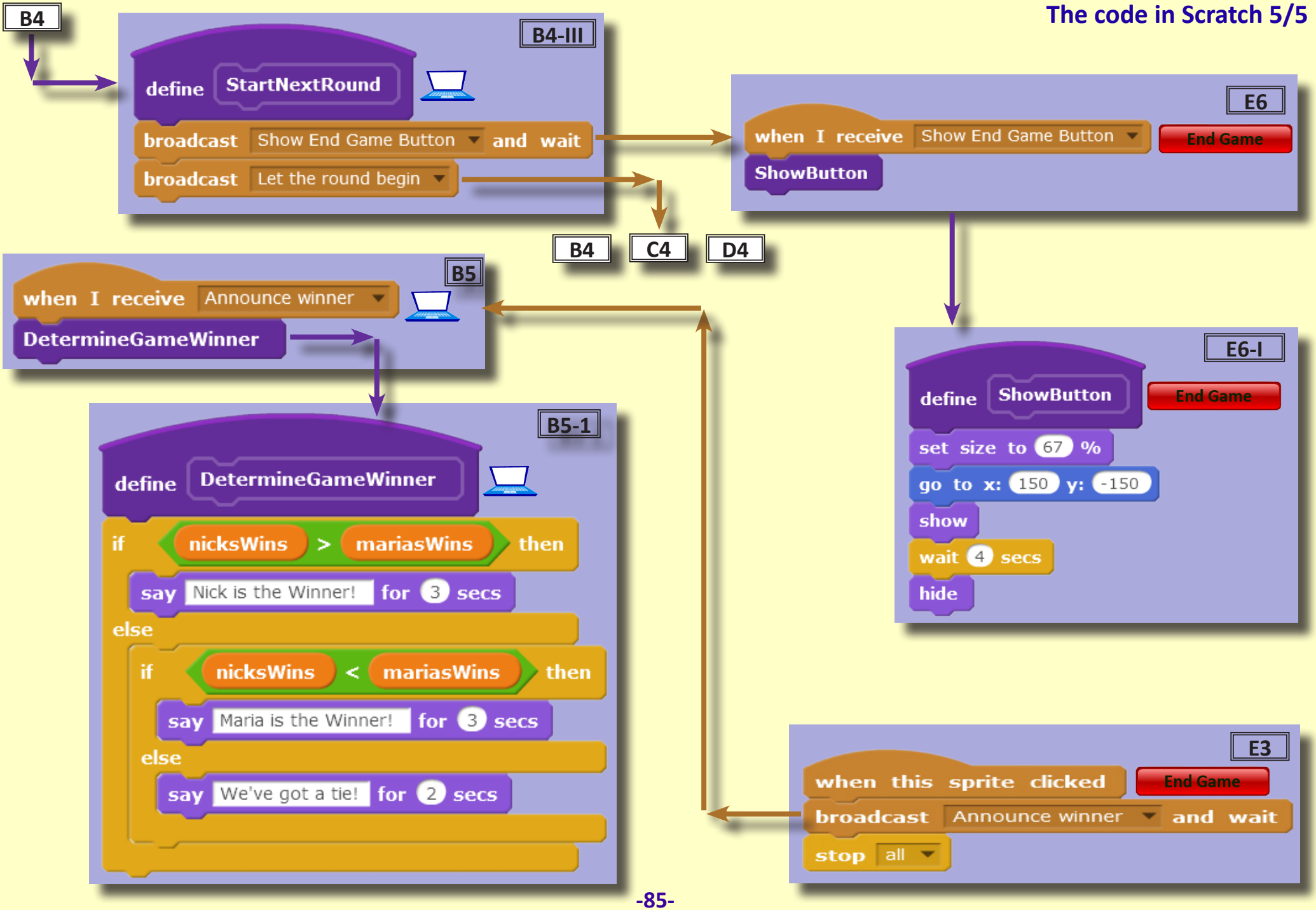
C4


C4-I
define PlayerSelects
set nickSelection to pick random 1 to 3
wait 1 secs
if nickSelection = Rock then
say Rock for 2 secs
else
if nickSelection = Scissors then
say Scissors for 2 secs
else
say Paper for 2 secs

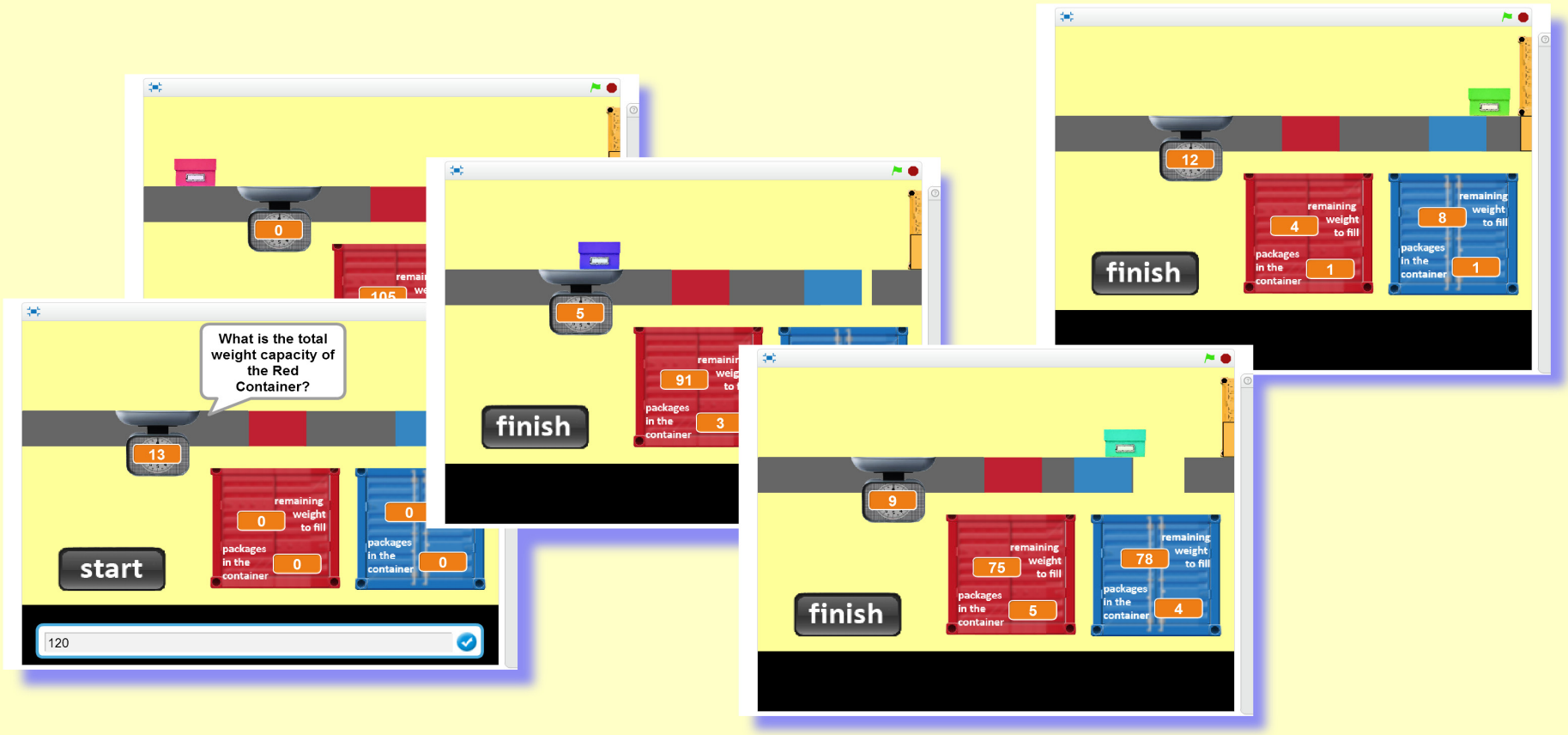
D4


D4-I
define PlayerSelects
set mariaSelection to pick random 1 to 3
wait 1 secs
if mariaSelection = Rock then
say Rock for 2 secs
else
if mariaSelection = Scissors then
say Scissors for 2 secs
else
say Paper for 2 secs

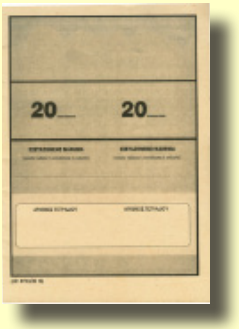

B4
B4-II
define RoundActions
if nickSelection = mariaSelection then
say Tie! for 2 secs
else
DetermineRoundWinner
change numberOfRounds by 1
B4-II-α
define DetermineRoundWinner
if nickSelection = Rock then
if mariaSelection = Scissors then
change nicksWins by 1
say Nick wins this round! for 2 secs
else
change mariasWins by 1
say Maria wins this round! for 2 secs
if nickSelection = Scissors then
if mariaSelection = Paper then
change nicksWins by 1
say Nick wins this round! for 2 secs
else
change mariasWins by 1
say Maria wins this round! for 2 secs
if nickSelection = Paper then
if mariaSelection = Rock then
change nicksWins by 1
say Nick wins this round! for 2 secs
else
change mariasWins by 1
say Maria wins this round! for 2 secs


B4
B4-III
define StartNextRound
broadcast Show End Game Button and wait
broadcast Let the round begin
B4
C4
D4
E6
when I receive Show End Game Button
End Game
ShowButton
B5
when I receive Announce winner
DetermineGameWinner
E6-I
define ShowButton
End Game
set size to 67 %
go to x: 150 y: -150
show
wait 4 secs
hide
B5-1
define DetermineGameWinner
if nicksWins > mariasWins then
say Nick is the Winner! for 3 secs
else
if nicksWins < mariasWins then
say Maria is the Winner! for 3 secs
else
say We've got a tie! for 2 secs
E3
when this sprite clicked
End Game
broadcast Announce winner and wait
stop all

6. Package router



Hellenic National Examinations Question



Secondary School Leaving Examination
May27th 2015
Tested course: Developing applications in a programming environment

Question 3

A transportation company has two warehouses, A and B, in the airport. During parcel receipts, every parcel is placed in the warehouse that currently has the most free space. If the free space of warehouse A equals that of warehouse B, the parcel is placed in warehouse A, but if the parcel doesn't fit in any of the two warehouses, it is forwarded to the company's main storage area outside the airport.

1. Write a program that will:
    a. Include an appropriate declare section.
    b. Read the free space sizesfor warehouses A and B.
    c. Read the size of each incoming parcel and subsequently display the name of the warehouse (A or B) where it will be stored, or alternatively, if the parcel doesn't fit in any of the warehouses, display the message "Parcel is being forwarded". The receipt process will be terminated when the user enters a parcel size of 0.
    d. Subsequently call a procedure (subprogram) that should find and display the name of the warehouse (A or B) which stores the most parcels, or display the message "Equal numbers of parcels" in the case of the two warehouses having stored the same number of parcels, or alternatively the message "No airport storage", if no parcel has been stored in any of the two warehouses A or B.

2. Write the procedure described in question 1d.

Hellenic National Examinations Question - Tested course: Developing Applications in a Programming Environment - Scratch adaptation

A transportation company stores the Packages  it will be transporting into two Containers  in order to send them to their destination..

Each Package is being weighed by a clever Scale-Router  .

Once the weighing is done, the Router forwards the Package to its proper storage area.

Each Package, its weight being between 5 and 15 kilos, is being loaded into that Container that can currently store the greatest load. If a Package cannot be loaded in either Container, then it is forwarded to the company warehouse.

Examples:
a. If an incoming package weighs 10 kilos, and the current weight capacity of the Red and the Blue containers is 50 and 30 kilos respectively, then the package will be loaded into the Red Container, and subsequently the Red Container's new weight capacity will be 40 kilos.

b. If the incoming package weighs 14 kilos, and the current weight capacity of the Red and the Blue containers is 12 and 10 kilos respectively, then the package will be forwarded to the warehouse.

These are the functional requirements of our program:

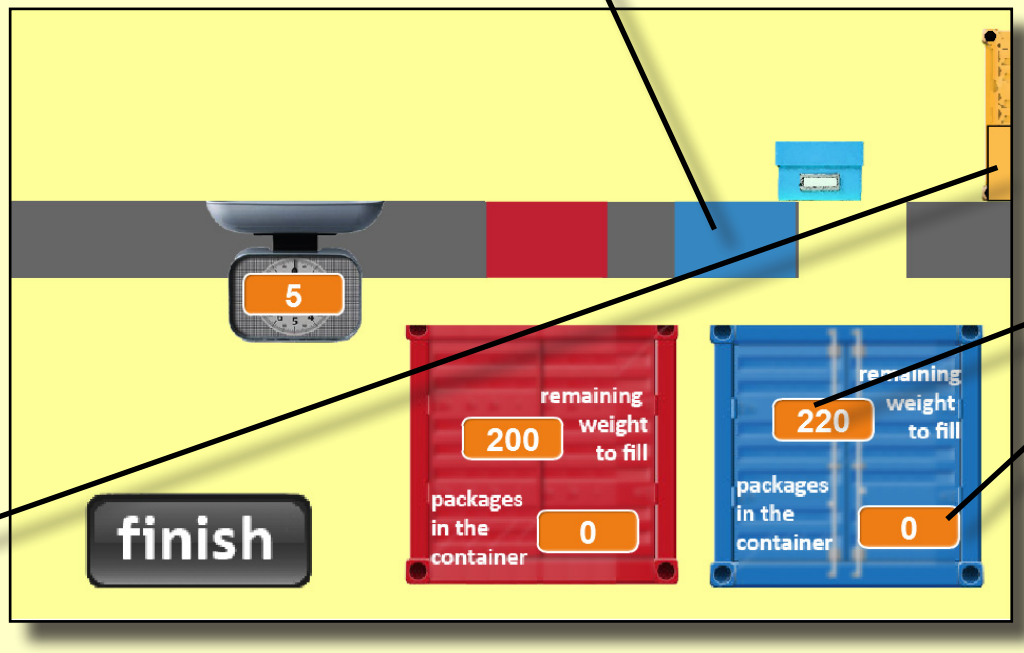
a. When the program starts (when the user clicks the green flag) the Start Button should show start .

b. When the user clicks the Start Button, the Scale-Router should ask the user to type in the initial capacity for each of the two Containers. After that, the Start Button shouldhide and the Finish Button should show up in the position of the Start Button finish .

c. The Scale-Router shouldbe taking care of forwarding thepackages either to one of the two Containers, or to the warehouse by sliding the corresponding slider open.

d. The package loading process should stop in one of two cases: if either each Container's capacity is less than 5 kilos, or the Finish Button is clicked.

e. On each Container two variables should appear:
its current capacity and
the number of packages being currently stored in it.



Program Control Flow

1. When the green flagis clicked by the user, the Start button scenario is activated making the button visible and initializing to 0 the counter variable startCheck.

2. Nothing happens until the user clicks the Start button.

3. When the Start Button is being clicked the message "Get Ready" is broadcast.

4. That message (Get Ready) is subsequently received by all the other sprites (8 in total):
    a. In all of the recipient sprites the scenario handling the message first shows the sprite, and then increments by 1 the startCheck variable (so that after all these scenarios have been executed this variable will be holding the value 8).

    b. Moreover, the Router sprite asks the user to enter the maximum weights the red and the blue containers can hold. These values are stored in the remainingWeightToFillRedContainer and remainingWeightToFillBlueContainer variables respectively.

    c. Additionally, the Red Container initializes to 0 the values of the remainingWeightToFillRedContainer accumulator and the redContainerNumberOfPackages counter.

    d. Accordingly, the Blue Container initializes to 0 the values of the remainingWeightToFillBlueContainer accumulator and the blueContainerNumberOfPackages counter.

    e. Furthermore, the FinishButton remains hidden until the other seven scenarios are completed and then shows in the same position as that of the Start Button.

5. Meanwhile, the Start Button waits for the startCheck variable to get the value 8. Once that happens (meaning that all other scenarios are done executing), then newPackageAllowed is raised, the "Next package please" message is broadcast, and the button is hidden so the user cannot click it to create another package before the current one has been routed.

Program Control Flow

6. When the Finish Button is being clicked, it broadcasts the "Terminate operation" message that subsequently receives to hide itself.

7. When the Router receives the "Next package please" message, it sets the incoming packageWeight to 0.

8. When the Package receives the "Next package please" message, it creates of clone of itself after checking that a new package is allowed.

9. Upon clone creation, the package appears at the start of its route, gets a random color and starts moving towards the router. When it reaches the router, it informs it by broadcasting the "Router please weigh me" message.

10. The Router, upon receiving the "Router please weigh me" message, "weighs" the package (by assigning a random value between 5 and 15), and then begins the package routing process:

a. During that process:

If the package doesn't fit in any of the containers

then it is forwarded to the warehouse.

Otherwise, the Router determines which of the two containers has the greatest remaining weight capacity and forwards accordingly the package to that container.

After the package-clone forwarding process is completed (targeting either the warehouse or one of the two containers), the clone is deleted.

Finally, a CheckTerminatingApplication is performed (i.e., whether the remaining weight capacity of each container is less than 5 kilos), and in that case the "Terminate operation" message is broadcast.

b. If the program is not terminated by the user, the "Next package please" message is broadcast at that point, and is subsequently received by the Router and Package sprites that repeat the above steps 7 and below).

Sprites' Functional Analysis

Start Button

When the user clicks the green flag

- The button shows and the startCheck counter-variable is initialized to 0.

The program enters a waiting state until the user clicks the button.

When the button is being clicked:

- The "Get Ready" message is broadcast and then the scenario waits until all sprites receiving the message are ready (until the startCheck variable gets the value 8).
- When the startCheck variable gets the value 8, then newPackageAllowed is raised (set to 1), the "Next package please" message is broadcast, and the button is hidden so the user cannot click it to create another package before the current one has been routed.

Finish Button

When the "Get Ready" message is being received:

- It remains hidden until all other sprites' show scenarios are finished and then shows (in the same position as that of the Start Button).
- It updates the Start Button that it is ready to start by incrementing by 1 the startCheck variable.

When it is being clicked:

- It broadcasts the message "Terminating operation" that it subsequently receives to hide itself.

Package

When the "Get Ready" message is being received:

- It shows and then informs the Start Button that it's ready to start by incrementing by 1 the start-Check variable.

When it receives the "Next package please" message:

- It checks whether a new package is allowed (newPackageAllowed=1) and then creates a clone of itself.

When a clone is being created

- The cloned package appears at the start of its path,
- gets a random color and
- moves toward the Router.
- When it reaches the Router, it sends the message "Router please weigh me".
- The clone is deleted after it has been forwarded to either the warehouse, the blue container or the red one.

Sprites' Functional Analysis

Router

When it receives the "Get Ready" message:

- It asks the user to enter the maximum weights that the red and blue containers can hold.
- It shows and notifies the Start Button that it's ready to start by incrementing the startCheck variable by 1.

When it receives the "Next package please" message:

- It initializes the upcoming package weight to 0.

When it receives the "Router please weigh me" message:

- It "weighs" it (assigning to the packageWeight variable a random value in the range 5-15), and then
- It begins the package routing process, whereby:

```
If the package doesn't fit in any of the containers
    then
        it is forwarded to the warehouse,
    otherwise,
        a check is done as to which of the two containers has the
        greatest remaining weight capacity and then the package is
        forwarded accordingly to that container.
```

Subsequently, a CheckTerminatingApplication is performed (i.e., whether the remaining weight capacity of each container is less than 5 kilos), and in that case the "Terminate operation" message is broadcast.

In the case the program is not terminated by the above check, the "Next package please" message is broadcast.

Sprites' Functional Analysis

Red Container

When it receives the "Get Ready" message:

- It initializes to 0 the values of the remainingWeightToFillRedContainer accumulator and the redContainerNumberOfPackages counter.
- It shows and notifies the Start Button that it's ready to start by incrementing the start-Check variable by 1.

Red Slider

When it receives the "Get Ready" message:

- It shows and notifies the Start Button that it's ready to start by incrementing the start-Check variable by 1.

Blue Container

When it receives the "Get Ready" message:

- It initializes to 0 the values of the remainingWeightToFillBlueContainer accumulator and the blueContainerNumberOfPackages counter.
- It shows and notifies the Start Button that it's ready to start by incrementing the start-Check variable by 1.

Blue Slider

When it receives the "Get Ready" message:

- It shows and notifies the Start Button that it's ready to start by incrementing the start-Check variable by 1.

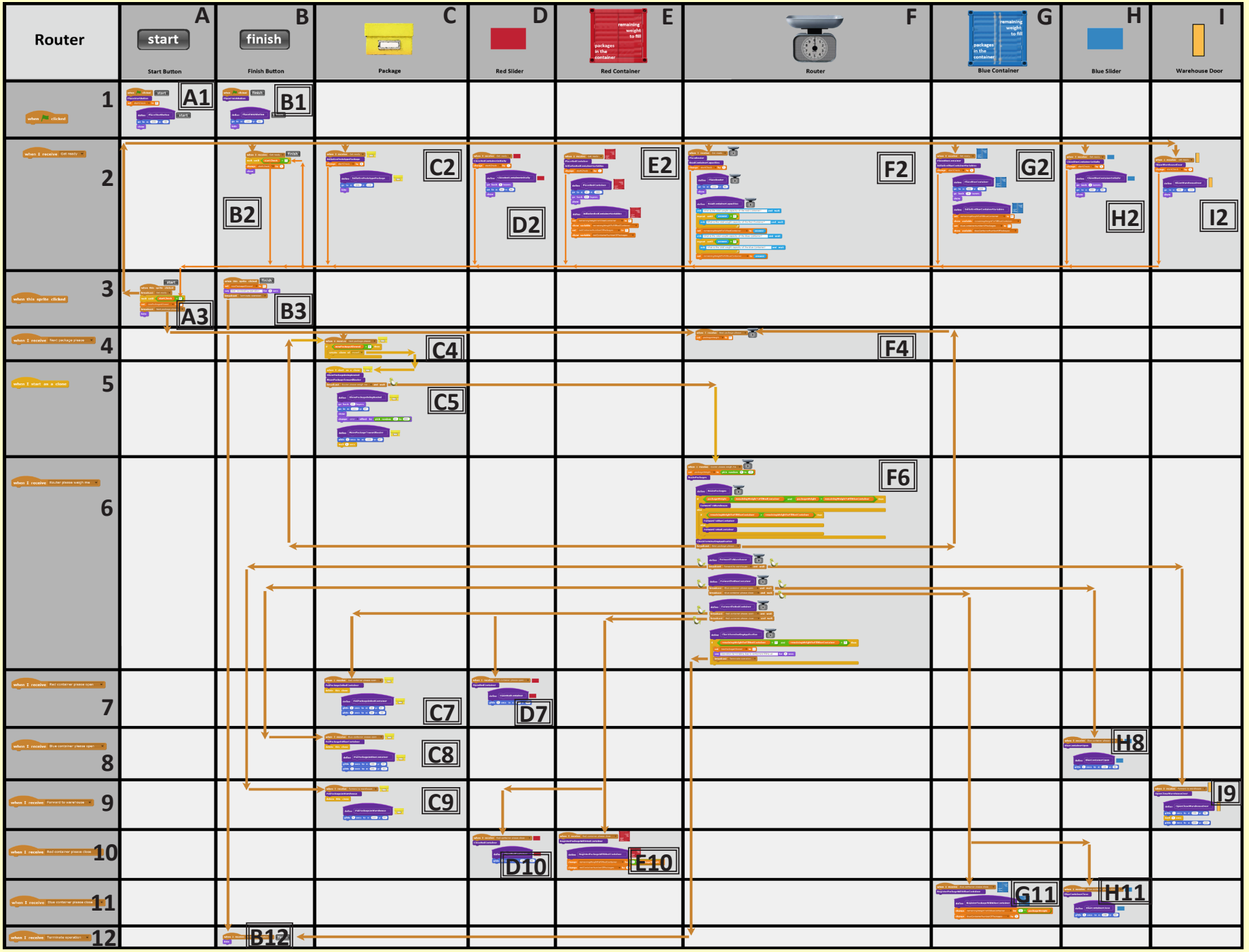
Warehouse Door

When it receives the "Get Ready" message:

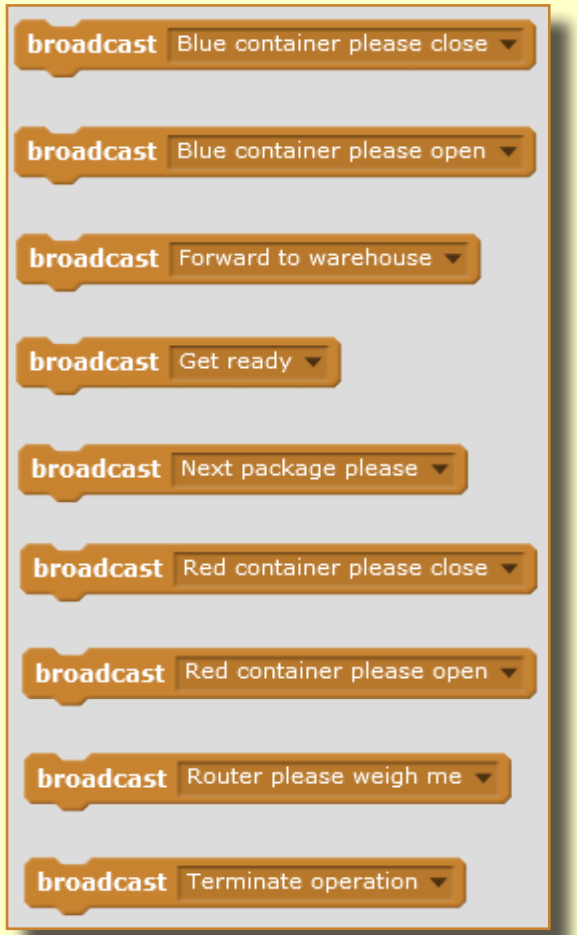
- It shows and notifies the Start Button that it's ready to start by incrementing the start-Check variable by 1.

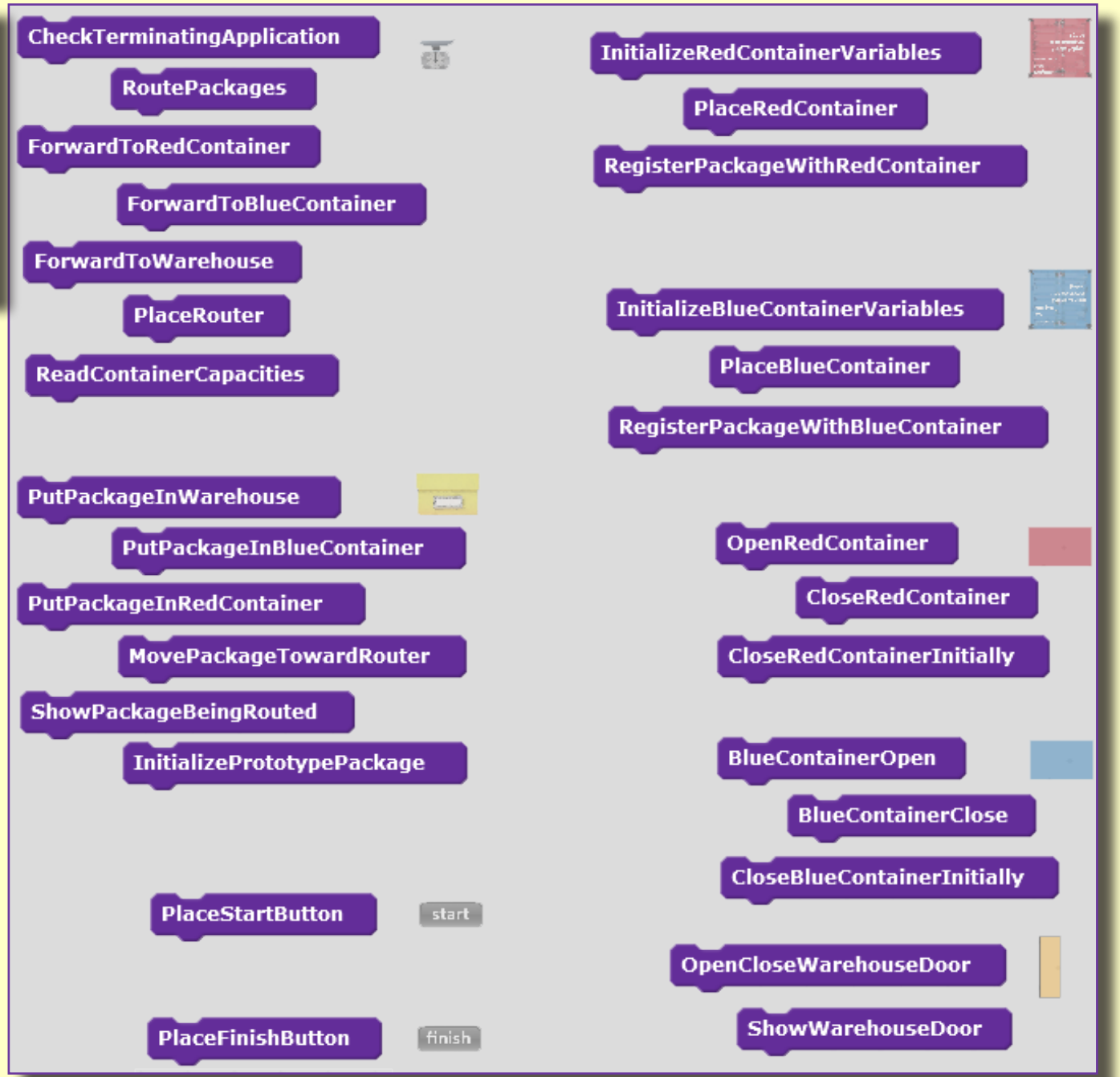
CodOrama

The table on this page, where we can see the various code modules and their interconnections, may be used in order to understand these scenarios.



Variables - messages - procedures

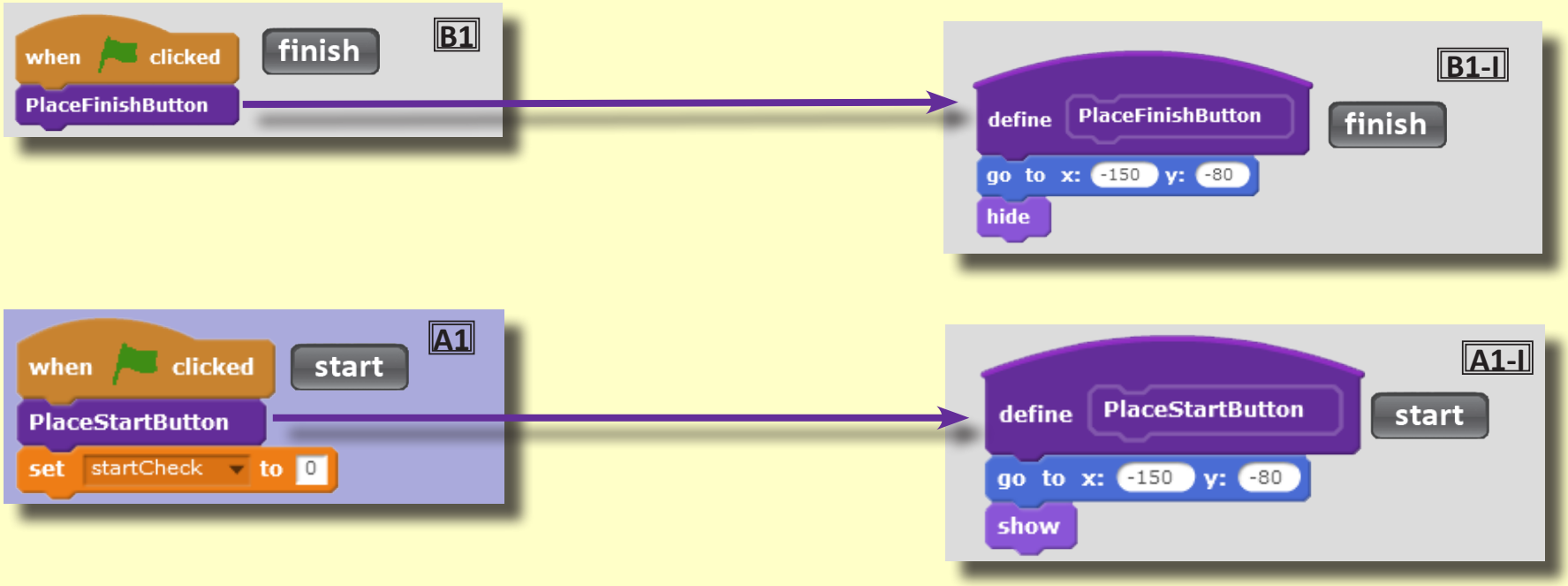


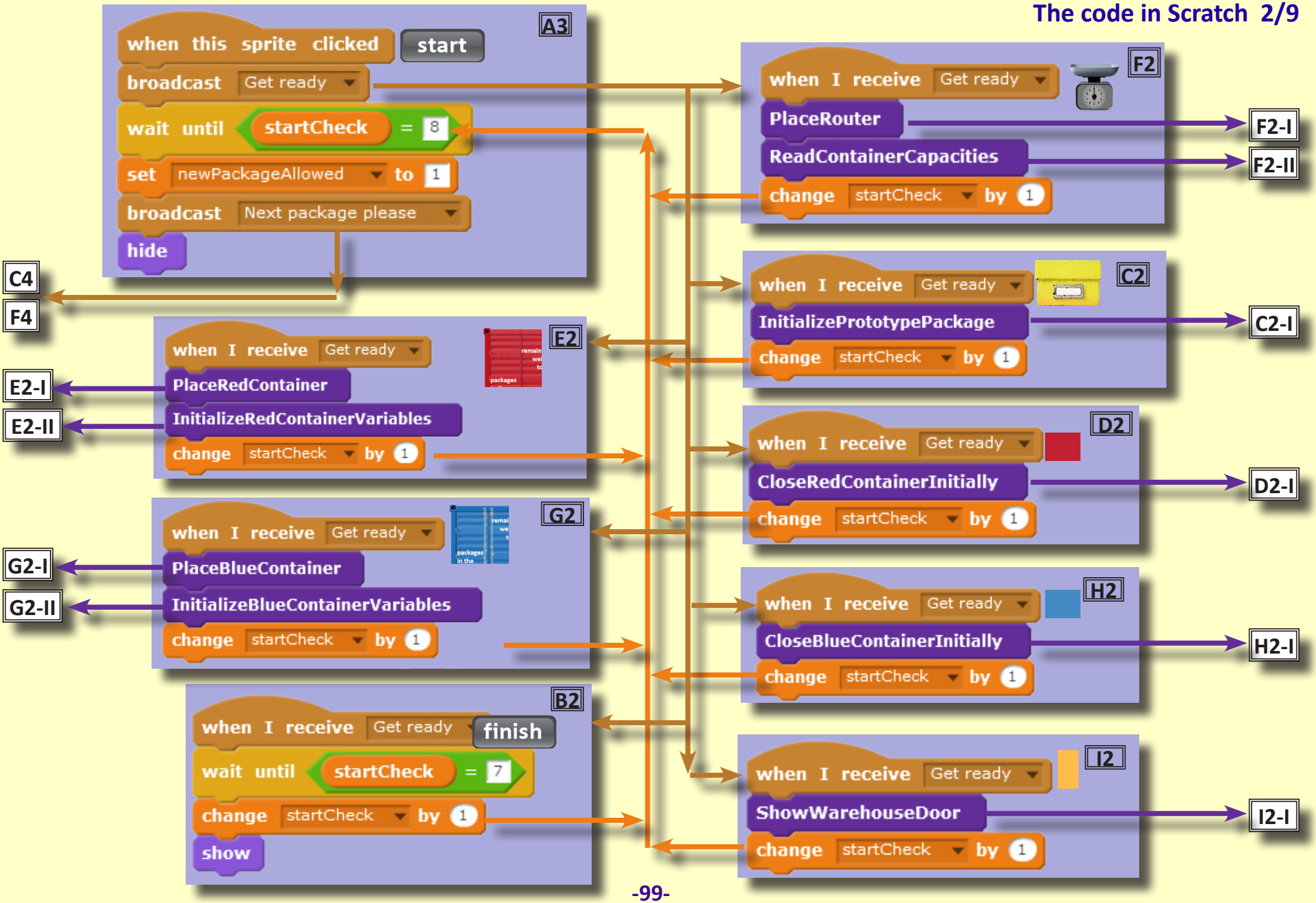


The code in Scratch

CodOrama: https://prezi.com/accewh_2lqyb/package-router/ or http://bit.ly/1KjWJjQ

Code: https://scratch.mit.edu/projects/82672224/ or http://bit.ly/1Lwai3u


when clicked
finish
B1
PlaceFinishButton
define PlaceFinishButton
finish
B1-I
go to x: -150 y: -80
hide
when clicked
start
A1
PlaceStartButton
set startCheck to 0
define PlaceStartButton
start
A1-I
go to x: -150 y: -80
show


A3
when this sprite clicked
start
broadcast Get ready
wait until startCheck = 8
set newPackageAllowed to 1
broadcast Next package please
hide
C4
F4
F2
when I receive Get ready
PlaceRouter
ReadContainerCapacities
change startCheck by 1
F2-I
F2-II
C2
when I receive Get ready
InitializePrototypePackage
change startCheck by 1
C2-I
E2
when I receive Get ready
PlaceRedContainer
InitializeRedContainerVariables
change startCheck by 1
E2-I
E2-II
D2
when I receive Get ready
CloseRedContainerInitially
change startCheck by 1
D2-I
G2
when I receive Get ready
PlaceBlueContainer
InitializeBlueContainerVariables
change startCheck by 1
G2-I
G2-II
H2
when I receive Get ready
CloseBlueContainerInitially
change startCheck by 1
H2-I
B2
when I receive Get ready
finish
wait until startCheck = 7
change startCheck by 1
show
I2
when I receive Get ready
ShowWarehouseDoor
change startCheck by 1
I2-I

C2

C2-I

```
define InitializePrototypePackage
go to x: -231 y: 110
hide
```

G2

G2-I

```
define PlaceBlueContainer
go to x: 162 y: -100
go back 10 layers
show
```

D2

D2-I

```
define CloseRedContainerInitially
go back 2 layers
go to x: 18 y: 86
show
```

H2

H2-I

```
define CloseBlueContainerInitially
go back 2 layers
go to x: 166 y: 86
show
```

E2

E2-I

```
define PlaceRedContainer
go to x: 17 y: -100
go back 10 layers
show
```

I2

I2-I

```
define ShowWarehouseDoor
go to x: 258 y: 123
show
```

E2

E2-II

```
define InitializeRedContainerVariables
set remainingWeightToFillRedContainer to 0
show variable remainingWeightToFillRedContainer
set redContainerNumberOfPackages to 0
show variable redContainerNumberOfPackages
```

G2

G2-II

```
define InitializeBlueContainerVariables
set remainingWeightToFillBlueContainer to 0
show variable remainingWeightToFillBlueContainer
set blueContainerNumberOfPackages to 0
show variable blueContainerNumberOfPackages
```

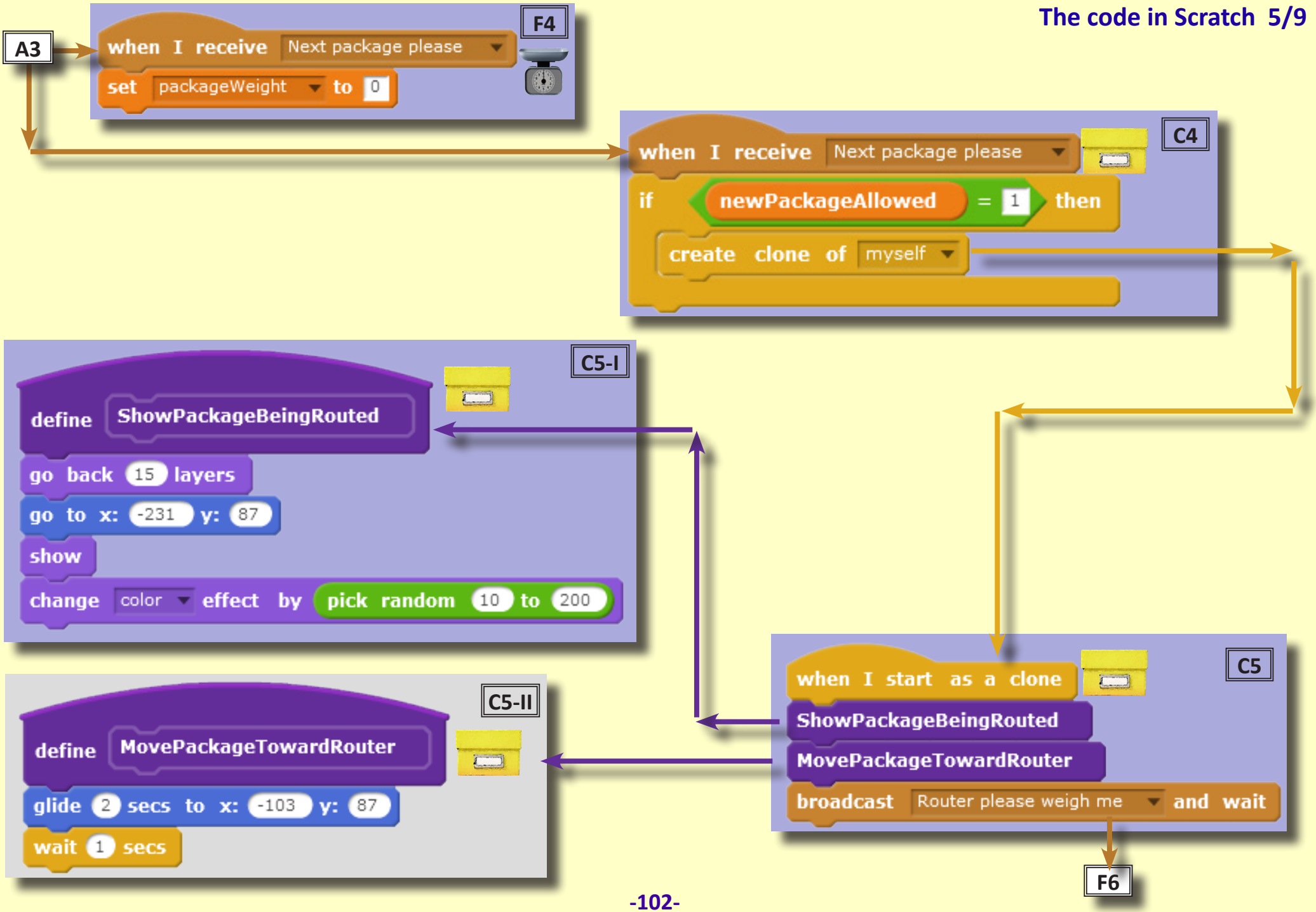
F2

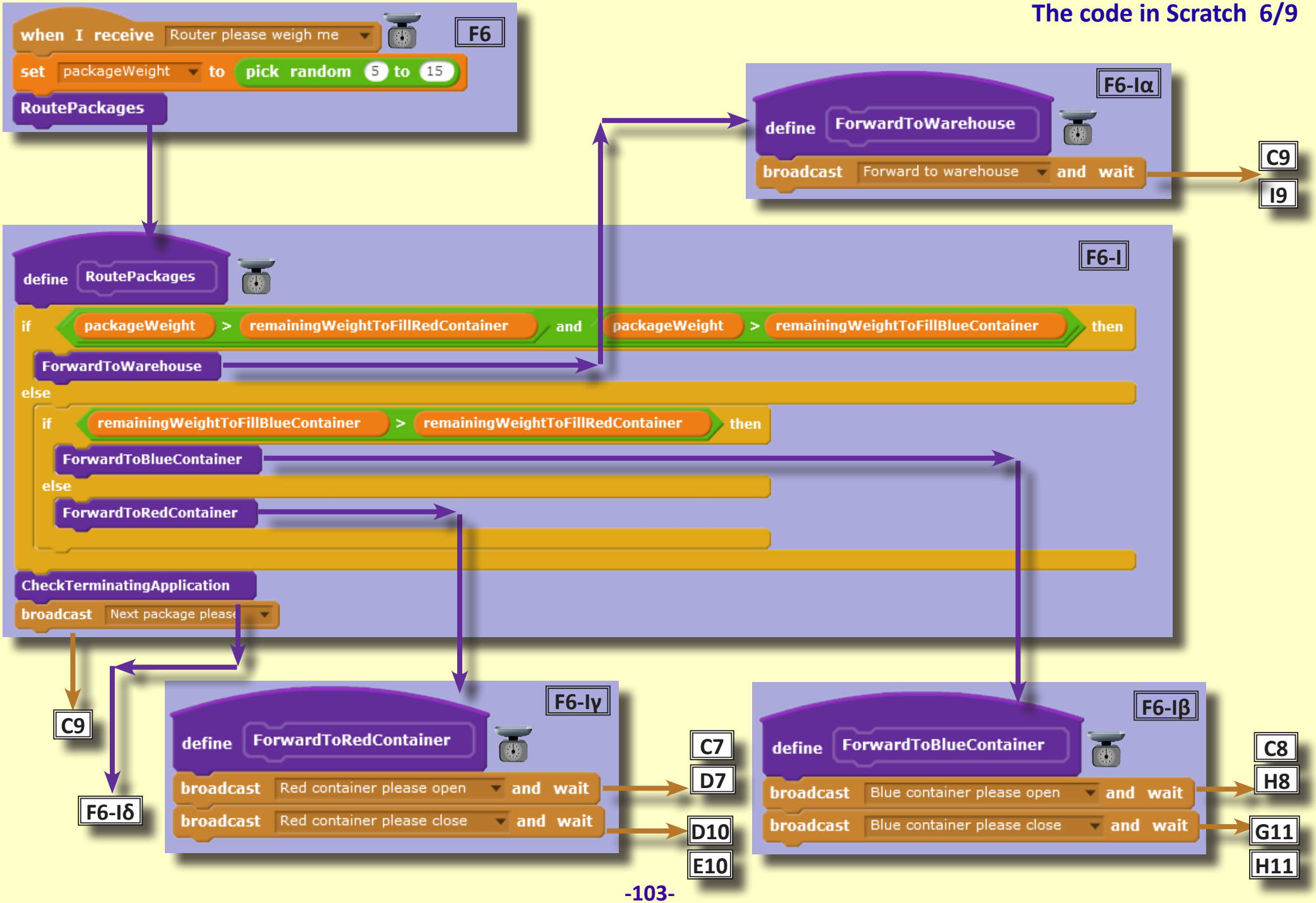
F2-I

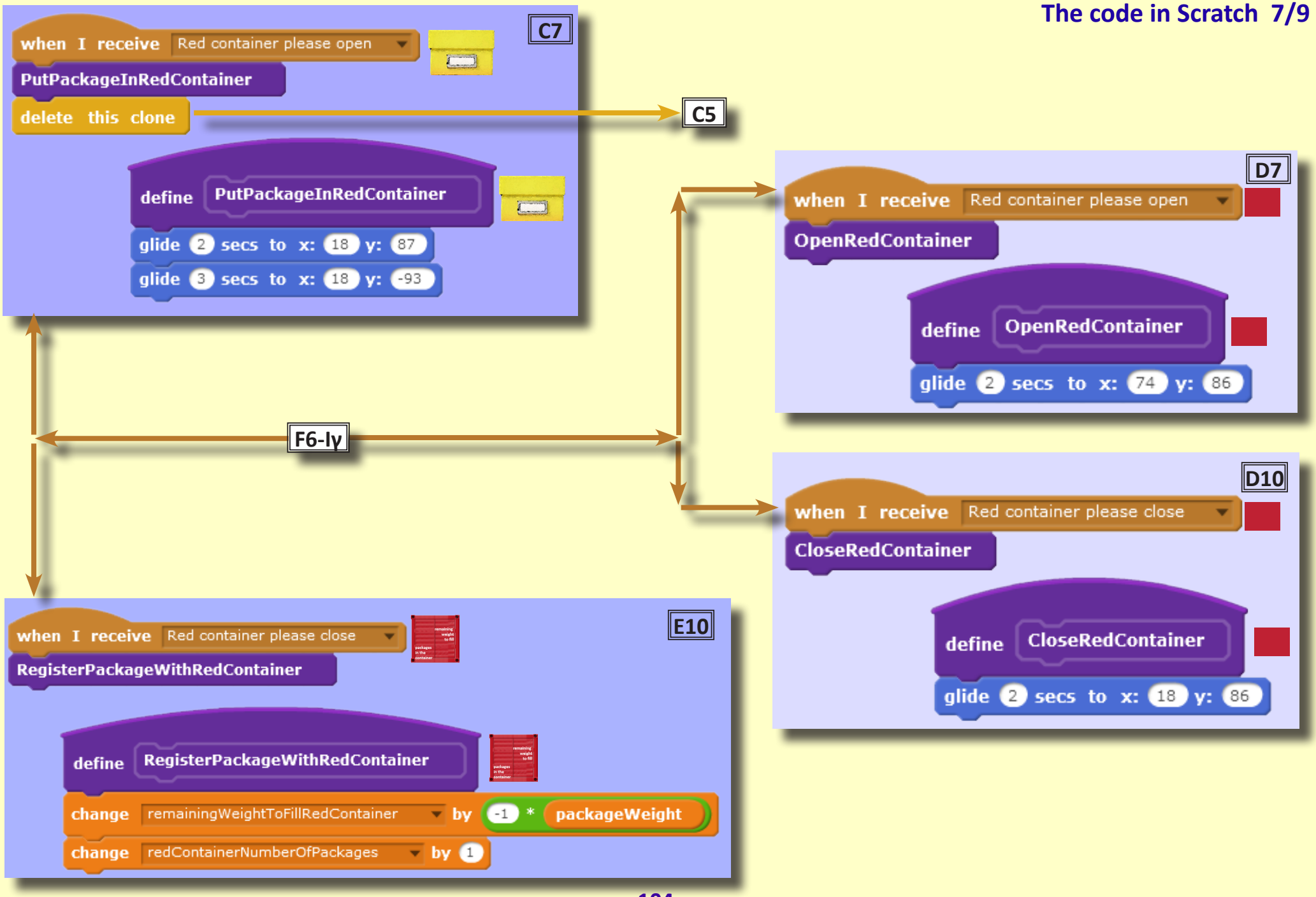
```
define PlaceRouter
go to x: -104 y: 86
show
```

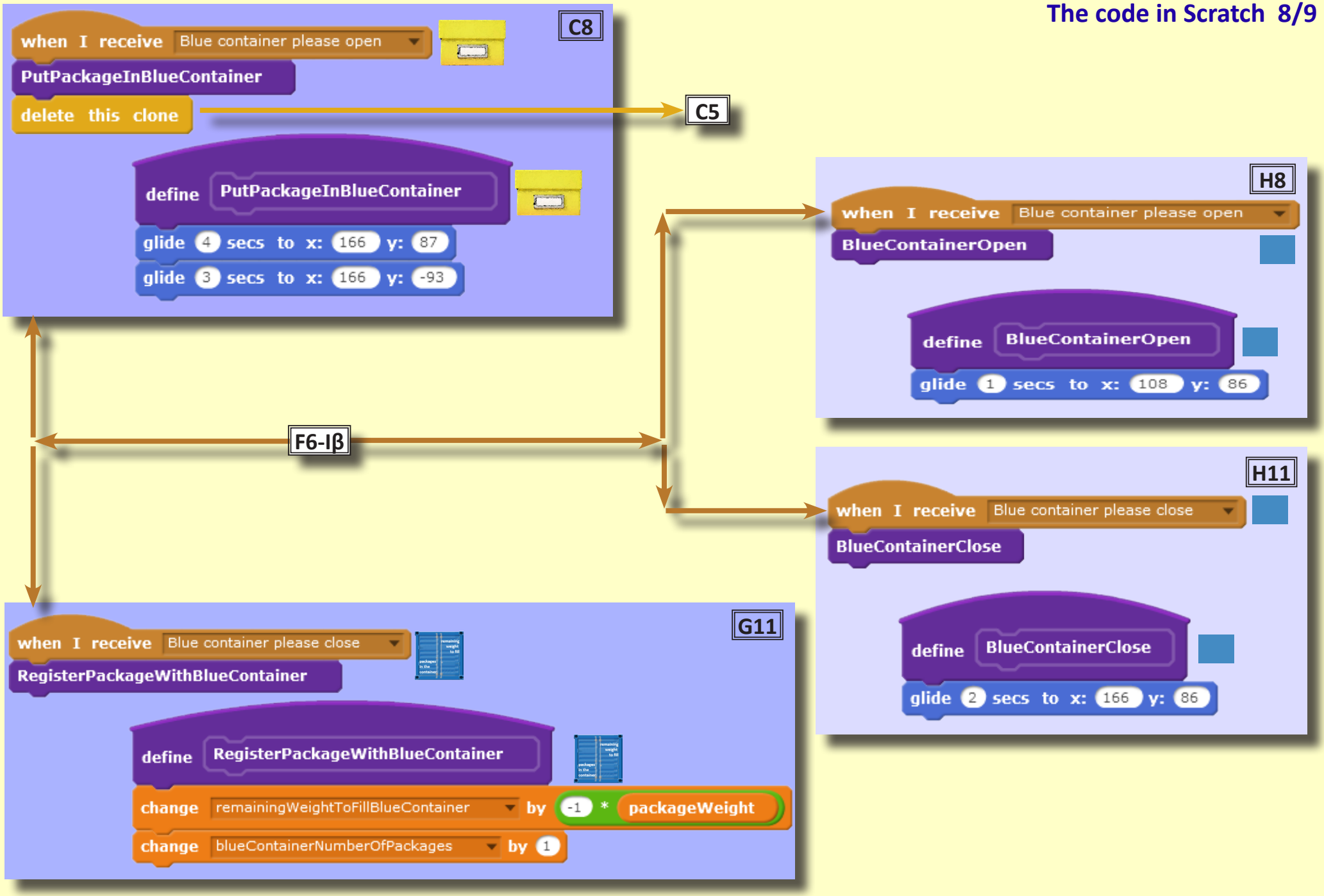
F2-II

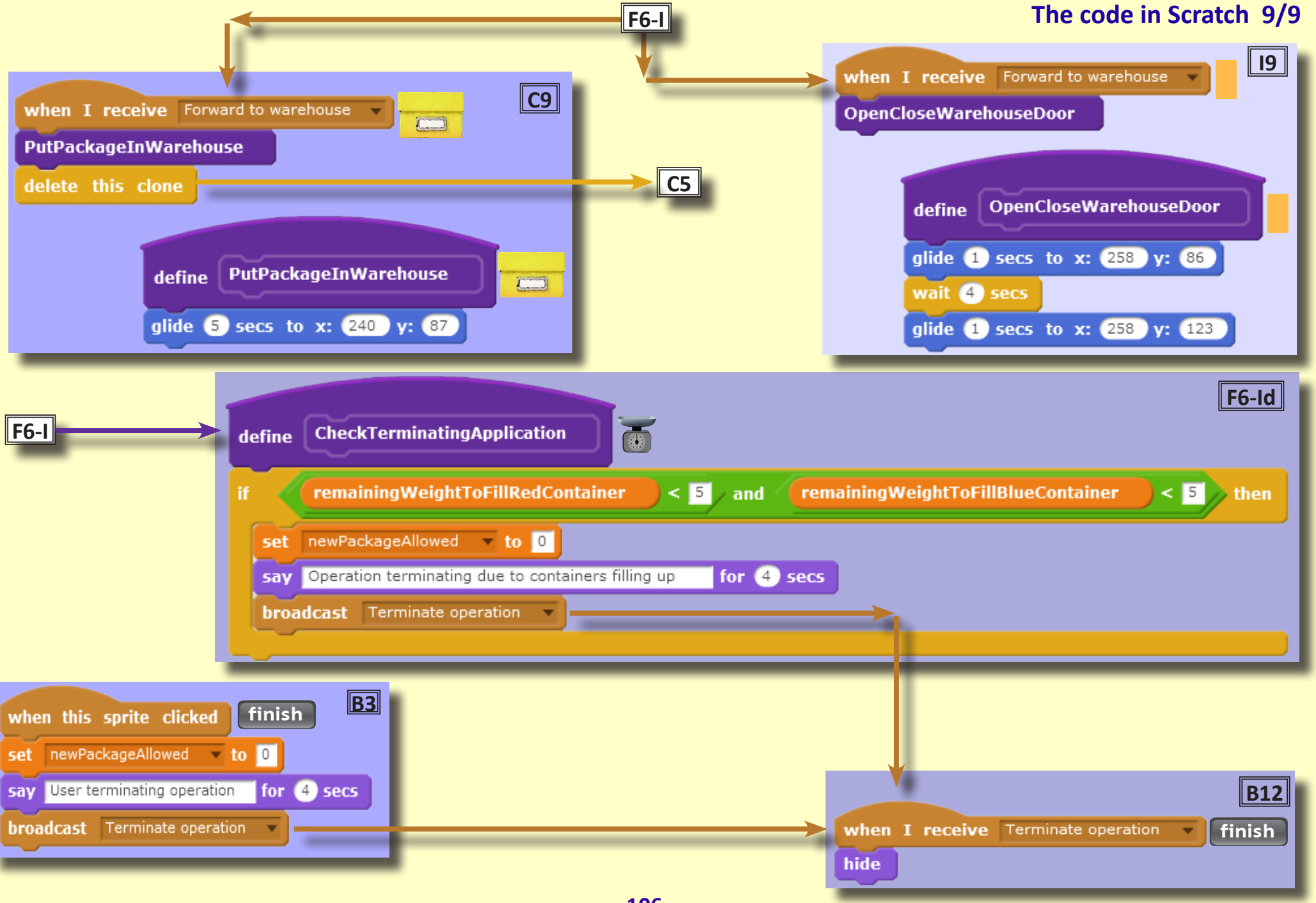
```
define ReadContainerCapacities
ask What is the total weight capacity of the Red Container? and wait
repeat until answer > 0
    ask What is the total weight capacity of the Red Container? and wait
set remainingWeightToFillRedContainer to answer
ask What is the total weight capacity of the Blue Container? and wait
repeat until answer > 0
    ask What is the total weight capacity of the Blue Container? and wait
set remainingWeightToFillBlueContainer to answer
```


A3
when I receive Next package please
set packageWeight to 0
F4
when I receive Next package please
if newPackageAllowed = 1 then
create clone of myself
C4
define ShowPackageBeingRouted
go back 15 layers
go to x: -231 y: 87
show
change color effect by pick random 10 to 200
C5-I
define MovePackageTowardRouter
glide 2 secs to x: -103 y: 87
wait 1 secs
C5-II
when I start as a clone
ShowPackageBeingRouted
MovePackageTowardRouter
broadcast Router please weigh me and wait
C5
F6


when I receive Router please weigh me
F6
set packageWeight to pick random 5 to 15
RoutePackages
F6-Iα
define ForwardToWarehouse
broadcast Forward to warehouse and wait
C9
I9
F6-I
define RoutePackages
if packageWeight > remainingWeightToFillRedContainer and packageWeight > remainingWeightToFillBlueContainer then
ForwardToWarehouse
else
if remainingWeightToFillBlueContainer > remainingWeightToFillRedContainer then
ForwardToBlueContainer
else
ForwardToRedContainer
CheckTerminatingApplication
broadcast Next package please
C9
F6-Iδ
F6-Iγ
define ForwardToRedContainer
broadcast Red container please open and wait
broadcast Red container please close and wait
C7
D7
D10
E10
F6-Iβ
define ForwardToBlueContainer
broadcast Blue container please open and wait
broadcast Blue container please close and wait
C8
H8
G11
H11


C7
when I receive Red container please open
PutPackageInRedContainer
delete this clone
define PutPackageInRedContainer
glide 2 secs to x: 18 y: 87
glide 3 secs to x: 18 y: -93
C5
D7
when I receive Red container please open
OpenRedContainer
define OpenRedContainer
glide 2 secs to x: 74 y: 86
F6-Iy
D10
when I receive Red container please close
CloseRedContainer
define CloseRedContainer
glide 2 secs to x: 18 y: 86
E10
when I receive Red container please close
RegisterPackageWithRedContainer
define RegisterPackageWithRedContainer
change remainingWeightToFillRedContainer by -1 * packageWeight
change redContainerNumberOfPackages by 1


C8
when I receive Blue container please open
PutPackageInBlueContainer
delete this clone
C5
define PutPackageInBlueContainer
glide 4 secs to x: 166 y: 87
glide 3 secs to x: 166 y: -93
H8
when I receive Blue container please open
BlueContainerOpen
define BlueContainerOpen
glide 1 secs to x: 108 y: 86
F6-Iβ
H11
when I receive Blue container please close
BlueContainerClose
define BlueContainerClose
glide 2 secs to x: 166 y: 86
G11
when I receive Blue container please close
RegisterPackageWithBlueContainer
define RegisterPackageWithBlueContainer
change remainingWeightToFillBlueContainer by -1 * packageWeight
change blueContainerNumberOfPackages by 1


F6-I
when I receive Forward to warehouse
PutPackageInWarehouse
delete this clone
C9
C5
define PutPackageInWarehouse
glide 5 secs to x: 240 y: 87
when I receive Forward to warehouse
OpenCloseWarehouseDoor
I9
define OpenCloseWarehouseDoor
glide 1 secs to x: 258 y: 86
wait 4 secs
glide 1 secs to x: 258 y: 123
F6-I
define CheckTerminatingApplication
F6-Id
if remainingWeightToFillRedContainer < 5 and remainingWeightToFillBlueContainer < 5 then
set newPackageAllowed to 0
say Operation terminating due to containers filling up for 4 secs
broadcast Terminate operation
when this sprite clicked
finish
B3
set newPackageAllowed to 0
say User terminating operation for 4 secs
broadcast Terminate operation
when I receive Terminate operation
finish
hide
B12

Bibliography

- LEAD Project, ***Super Scratch Programming Adventure! (Covers Version 2): Learn to Program by Making Cool Games***, Starch Press; 2 edition (31 Oct. 2013)

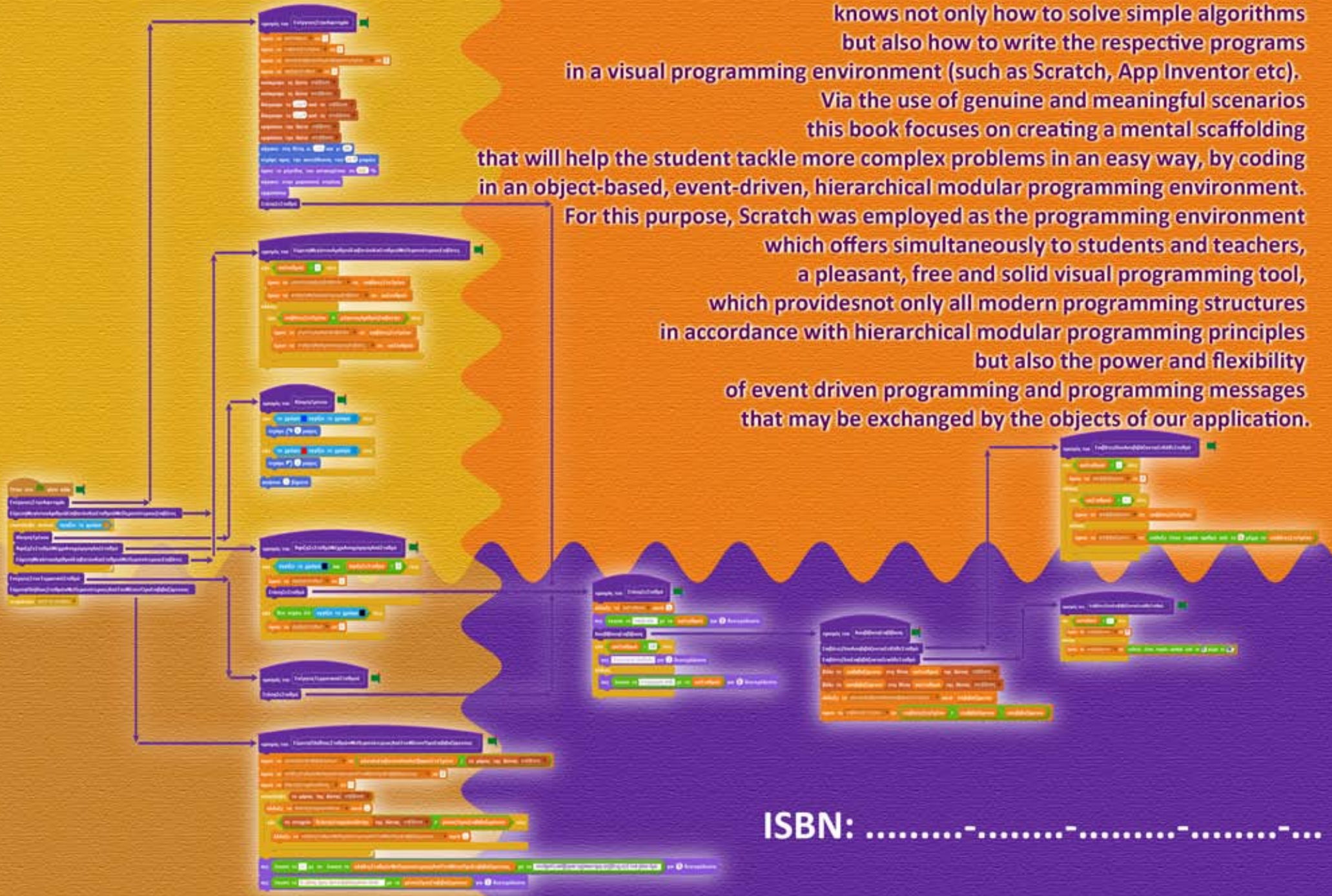
- Manataki Areti, Inés Friss de Kereki, **Code Yourself! An Introduction to Programming (Scratch)**, MOOC / Coursera, https://www.coursera.org/course/codeyourself, Mar 9th – Apr 12th 2015.

- Λαδιάς Τ., Τσιωτάκης Π., Φεσάκης Γ., ***Οδηγός Υποχρεωτικής Εκπαίδευσης, για τον εκπαιδευτικό στο επιστημονικό πεδίο: Πληροφορική και Νέες Τεχνολογίες***. Πράξη «ΝΕΟ ΣΧΟΛΕΙΟ (Σχολείο 21ου αιώνα) – Νέο πρόγραμμα σπουδών», Αξονες Προτεραιότητας 1,2,3 - Οριζόντια Πράξη. 2012.

- Παπαδόπουλος Γ., Λαδιάς Τ., Φωτιάδης Δ., ***Σενάρια Διδασκαλίας σε Περιβάλλοντα Οπτικού Προγραμματισμού με Πλακίδια***, http://bit.ly/1V2DrnZ (απαιτείται εγγραφή για την προσπέλαση του υλικού). 2014.

- Συγγραφική ομάδα του ScratchEd, ***Δημιουργική Χρήση Υπολογιστή με το Scratch***, (http://bit.ly/1FnkOsX)

It is our firm belief that the reader
knows not only how to solve simple algorithms
but also how to write the respective programs
in a visual programming environment (such as Scratch, App Inventor etc).
Via the use of genuine and meaningful scenarios
this book focuses on creating a mental scaffolding
that will help the student tackle more complex problems in an easy way, by coding
in an object-based, event-driven, hierarchical modular programming environment.
For this purpose, Scratch was employed as the programming environment
which offers simultaneously to students and teachers,
a pleasant, free and solid visual programming tool,
which providesnot only all modern programming structures
in accordance with hierarchical modular programming principles
but also the power and flexibility
of event driven programming and programming messages
that may be exchanged by the objects of our application.



ISBN: .....-.....-.....-.....-...