

ICSP

AppInventor
Lecture 3

Outline

- Lists
- Subroutines
- Communication protocols

Lists

- Abstract data structure that implements an ordered collection of values, the same value can occur more than once
- The size of the list indicates the number of elements
- Elements can be accessed via their index
- Elements can be concatenated to the list

List:

c	4	a	1
---	---	---	---

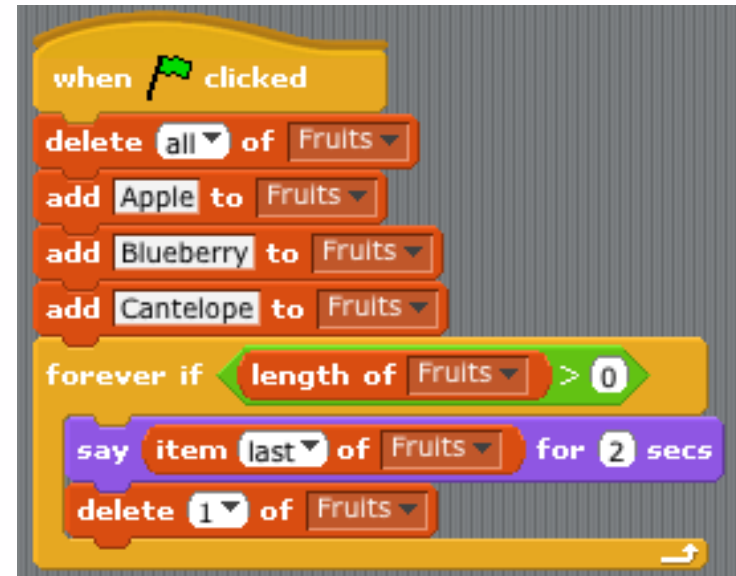
Index:

0	1	2	3
---	---	---	---



List example

- When the program starts fruits will be added to the fruit list.
- After the fruits are added the Sprite will, as long as there are fruits in the list, say the name of the last fruit in the list and then remove it from the list



Queues & Stacks

- Queues and Stacks are two versions of lists with different properties
- In a queue the first element that is added to the queue (enqueue) is the first to be taken from it (dequeue) Called First in, First Out (FIFO)
 - E.g. The checkout line in a supermarket is a queue
- In a stack the last element to be put(push) on the stack is the first to be taken off the top(popped) Called Last in, First Out (LIFO)
 - E.g. A pile of books can be described as a stack

Rock, Paper or Scissors

- Rules:
 - Rock beats Scissors
 - Scissors beats Paper
 - Paper beats Rock
- **Exercise**: Code a Rock, Paper or Scissors game in Scratch:
 - You make a choice of Rock, Paper or Scissors with the keys “r”, “p” or “s”
 - The computer makes a random choice Rock, Paper or Scissors
 - The result is displayed
- (Bonus if you use a list, e.g. The values of “rock”, “paper” or “scissors” are all player choices)

```
when I receive player_selected
  if ai_choice = player_choice
    say draw
  else
    if solutions contains join player_choice ai_choice
      say Win
    else
      say Lose
```

Compare ai and player choice
Equal choices means a draw

The player wins if the joined choices are in the solutions list

```
when clicked
  if length of choices < 3
    add rock to choices
    add paper to choices
    add scissors to choices
  if length of solutions < 3
    add rockscissors to solutions
    add paperrock to solutions
    add scissorspaper to solutions
  set ai_choice to item pick random 1 to length of choices of choices
  set player_choice to none
  say Select (r)ock, (p)aper or (s)ciissor
```

Add rock(1), paper(2) and scissors(3) to the choices. If there are less than 3 choices in the list

The ai selects a random number from the list. The players choice is cleared

```
when r key pressed
  set player_choice to item 1 of choices
  broadcast player_selected
```

r selects rock(1) from choices

```
when p key pressed
  set player_choice to item 2 of choices
  broadcast player_selected
```

p selects paper(2) from choices

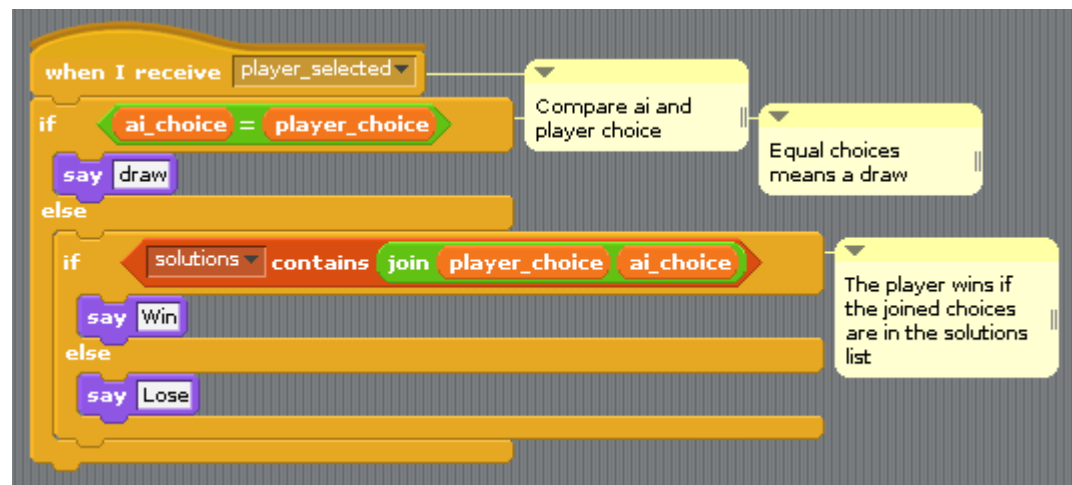
```
when s key pressed
  set player_choice to item 3 of choices
  broadcast player_selected
```

s selects scissors(3) from choices

Subroutine

A subroutine (procedure, function, routine, method, or subprogram)

- Portion of code within a larger program that performs a specific task and is somewhat independent piece of code.
 - _ Used to structure the code
 - _ Used when repeated calls need to be made
 - _ Reduces the cost of developing and maintaining a large program and increases its quality and reliability.
- Subroutines, collected into libraries, are an important mechanism for sharing software



AppInventor Rock, Paper and Scissors

- Create 3 buttons
 - Rock
 - Paper
 - Scissors
- Display the result of your choice against a random AI
- Think from Scratch

Palette

- Basic
- Media
- Animation
- Social
- Sensors
- Screen Arrangement**
 - HorizontalArrangement
 - TableArrangement
 - VerticalArrangement
- LEGO® MINDSTORMS®
- Other stuff
- Not ready for prime time
- Old stuff

Viewer

Display Invisible Components in Viewer

Screen1

Select Rock, Paper or Scissors

Rock

Paper

Scissors

Player:

AI:

Components

- Screen1
 - Instructions
 - Rock
 - Paper
 - Scissor
 - HorizontalArrangement1
 - player_lbl
 - player_selection
 - HorizontalArrangement2
 - AI_label
 - AI_selection
 - Result

Rename... Delete...

Properties

BackgroundColor
 None

FontBold

FontItalic

FontSize
14.0

FontTypeface
default

Text
[Empty text box]

TextAlignment
left

TextColor
Black

Visible

Width
Automatic...

Height
Automatic...

Media

Add...

```

when I receive player_selected
  if ai_choice = player_choice
    say draw
  else
    if solutions contains join player_choice ai_choice
      say Win
    else
      say Lose
  
```

Compare ai and player choice
Equal choices means a draw

The player wins if the joined choices are in the solutions list

```

when clicked
  if length of choices < 3
    add rock to choices
    add paper to choices
    add scissors to choices
  if length of solutions < 3
    add rockscissors to solutions
    add paperrock to solutions
    add scissorspaper to solutions
  set ai_choice to item pick random 1 to length of choices of choices
  set player_choice to none
  say Select (r)ock, (p)aper or (s)issor
  
```

Add rock(1), paper(2) and scissors(3) to the choices. If there are less than 3 choices in the list

The ai selects a random number from the list. The players choice is cleared

```

when r key pressed
  set player_choice to item 1 of choices
  broadcast player_selected
  
```

r selects rock(1) from choices

```

when p key pressed
  set player_choice to item 2 of choices
  broadcast player_selected
  
```

p selects paper(2) from choices

```

when s key pressed
  set player_choice to item 3 of choices
  broadcast player_selected
  
```

s selects scissors(3) from choices

```

to do_turn arg
  set global ai_choice to call pick random item list global choices
  set AI_selection.Text to global ai_choice
  set player_selection.Text to global player_choice
  ifelse test text1 global ai_choice text2 global player_choice text=
  then-do set Result.Text to text Draw
  else-do ifelse test call thing global player_choice join global ai_choice is in list? list global solutions
  then-do set Result.Text to text Win
  else-do set Result.Text to text Lose
  
```

```

def choices as call make a list
  item text rock
  item text paper
  item text scissors
  
```

```

def solutions as call make a list
  item text rock join text scissors
  item text paper join text rock
  item text scissors join text paper
  
```

```

def player_choice as text none
def ai_choice as text none
  
```

```

when Rock.Click
  do set global player_choice to call select list item list global choices index number 1
  call do_turn
  
```

```

when Paper.Click
  do set global player_choice to call select list item list global choices index number 2
  call do_turn
  
```

```

when Scissor.Click
  do set global player_choice to call select list item list global choices index number 3
  call do_turn
  
```

```

def ai_choice as text none
  to do_turn arg
  do
    set global ai_choice to call pick random item list global choices
    set AI_selection.Text to global ai_choice
    set player_selection.Text to global player_choice
    ifelse test text1 global ai_choice
           text= text2 global player_choice
    then-do set Result.Text to text Draw
    else-do ifelse test call is in list? thing global player_choice join global ai_choice
            list global solutions
            then-do set Result.Text to text Win
            else-do set Result.Text to text Lose
  end
end

```

```

def player_choice as text none

```

```

def solutions as
  call make a list
  item text rock join text scissors
  item text paper join text rock
  item text scissors join text paper

```

```

def choices as
  call make a list
  item text rock
  item text paper
  item text scissors

```

```

when Scissor.Click
  do
    set global player_choice to call select list item list global choices
    index number 3
    call do_turn
  end

```

```

when Paper.Click
  do
    set global player_choice to call select list item list global choices
    index number 2
    call do_turn
  end

```

```

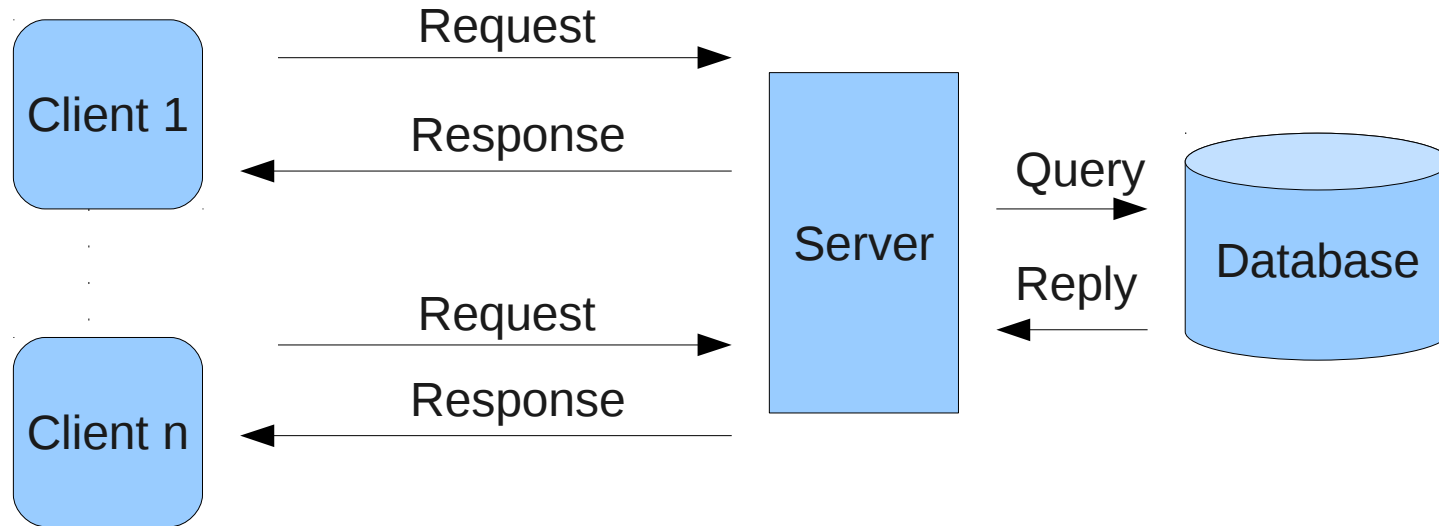
when Rock.Click
  do
    set global player_choice to call select list item list global choices
    index number 1
    call do_turn
  end

```

Between a rock, a paper or scissors

- A drama about communication in one act
- Scenen: A Rock, Paper, Scissors tournament
- Actors
 - Clients – Participates in the tournament
 - Name: Rick, likes to roll with Rock
 - Name: Piper, likes to fold with Paper
 - Name: Saussage, likes to run with Scissors
 - Server – Handles the Rock, Paper, Scissors tournament requests, responses and data storage
 - Gets players name and responds with player results
 - Responds with all stored player names if requested
 - Stores player name and move

Communication



Communication Protocols

- How do you make the devices (e.g. PC or Smartphone) communicate with each other in order to register Rock, Paper or Scissors players, moves and results?
 - **Excercise**: Write down in pseudocode how you would handle the communication

Rock, Paper or Scissors - Multiplayer

- The devices do not know that other devices exist. Therefore the webpage serves the clients(apps) and stores information.

- A webpage has been set up <http://icsp2011rps.appspot.com/>

1. Register a name on the device

1. If there is no name registered notify that a name is missing
2. Else show the options “Store Move” and “Compete against an opponent”

2. Store a move on the server in the registered name

1. Play a move
2. Store the move on the server's data base

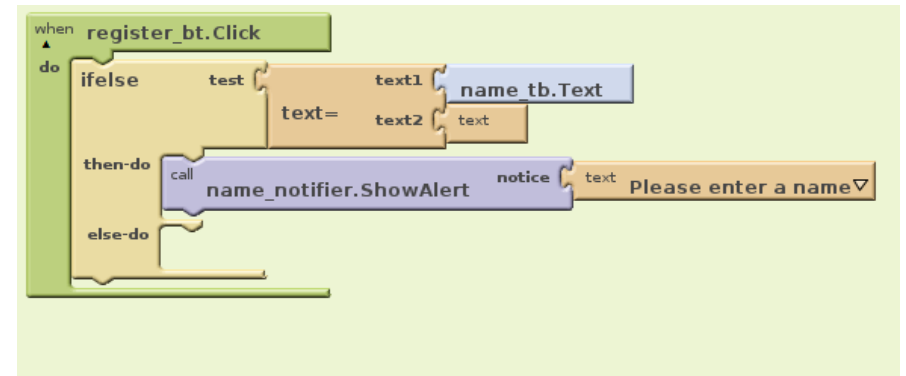
3. Compete against an opponent. The moves and opponents are stored on the server

1. Chose an opponent listed in the server's database
2. Play a move
3. Calculate the result
4. Show the result

Register a name

The screenshot shows the MIT App Inventor interface for a project named "rps_mult". The interface is divided into several panes:

- Palette:** A list of components on the left, including "ActivityStarter", "BarcodeScanner", "BluetoothClient", "BluetoothServer", "Notifier", "SpeechRecognizer", "TextToSpeech", "TinyWebDB", and "Web".
- Viewer:** A central window showing a mobile screen with a text input field and a "Register" button. The screen title is "Screen1".
- Components:** A pane on the right showing the components added to the screen: "name_tb", "register_bt", "icsp2011rpsDB", and "name_notifier".
- Properties:** A pane on the right for setting properties, which is currently empty.
- Bottom:** Buttons for "Rename...", "Delete...", and "Add..." are visible.



Chose to Store Move or Play Opponent

The screenshot displays the LEGO Mindstorms Blocks Editor interface for a project named "rps_mult". The interface is divided into several panels:

- Top Bar:** Contains "Save", "Save As", and "Checkpoint" buttons. On the right, it shows "Blocks Editor is open" and a "Package for Phone" dropdown menu.
- Palette:** A vertical sidebar on the left with categories: Basic, Media, Animation, Social, Sensors, Screen Arrangement, LEGO® MINDSTORMS®, Other stuff, Not ready for prime time, and Old stuff. Under "Screen Arrangement", three options are listed: HorizontalArrangement, TableArrangement, and VerticalArrangement.
- Viewer:** The central workspace showing a mobile screen titled "Screen1". The screen has a status bar at the top with icons for signal, battery, and time (5:09 PM). Below the status bar is a text input field and a "Register" button. A checkbox labeled "Display Invisible Components in Viewer" is checked. Below the viewer, a "Non-visible components" section shows a tree structure: "icsp2011rpsDB" (green arrow icon) and "name_notifier" (yellow triangle icon).
- Components:** A tree view on the right showing the hierarchy of components on the screen: "Screen1" (parent), "start_va" (child), "name_tb" (child of start_va), "register_bt" (child of start_va), "store_move_bt" (child of start_va), "show_opponents_lp" (child of start_va), "move_va" (child of start_va), "icsp2011rpsDB" (child of start_va), and "name_notifier" (child of start_va). The "move_va" component is highlighted in green. Below the tree are "Rename..." and "Delete..." buttons.
- Properties:** A panel on the far right showing the properties of the selected "move_va" component. It includes "Visible" (checkbox), "Width" (set to "Automatic..."), and "Height" (set to "Automatic...").
- Media:** A section at the bottom right with an "Add..." button.

Choose a Move

Palette

Basic

- Button
- Canvas
- CheckBox
- Clock
- Image
- Label
- ListPicker
- PasswordTextBox
- TextBox
- TinyDB

Media

Animation

Social

Sensors

Screen Arrangement

LEGO® MINDSTORMS®

Other stuff

Not ready for prime time

Old stuff

Viewer

Display Invisible Components in Viewer

Screen1

5:09 PM

Register

Make a choice

Rock

Paper

Scissors

Particitant Choice

	None
None	None
Result	None

Non-visible components

- icsp2011rpsDB
- name_notifier

Components

- register_bt
- store_move_bt
- show_opponents_lp
- move_va
 - choice_lbl
 - rock_bt
 - paper_bt
 - scissors_bt
- results_tbl
 - participant_lbl_r_tbl
 - move_lbl_r_tbl
 - player_name_lbl_r_tbl
 - player_move_lbl_r_tbl
 - opponent_name_lbl_r_tbl
 - opponent_move_lbl_r_tbl
 - result_lbl_r_tbl
 - result_type_lbl_r_tbl
- icsp2011rpsDB
- name_notifier

Rename... Delete...

Media

Add...

Properties

BackgroundColor

None

FontBold

FontItalic

FontSize

14.0

FontTypeface

default

Text

None

TextAlignment

left

TextColor

Black

Visible

Checked

Width

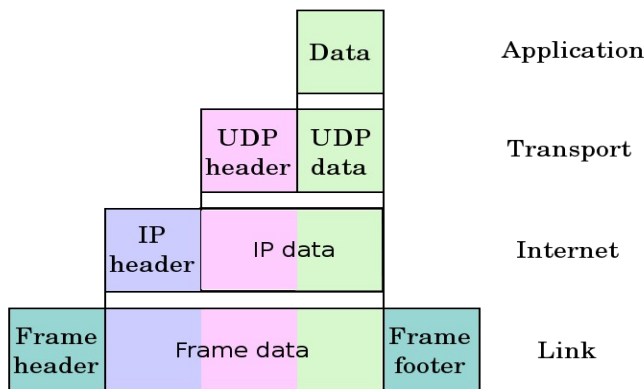
Automatic...

Height

Automatic...

Internet Protocol Stack

- The lowest protocol deals with physical interaction of the hardware. Every layer above adds more features.



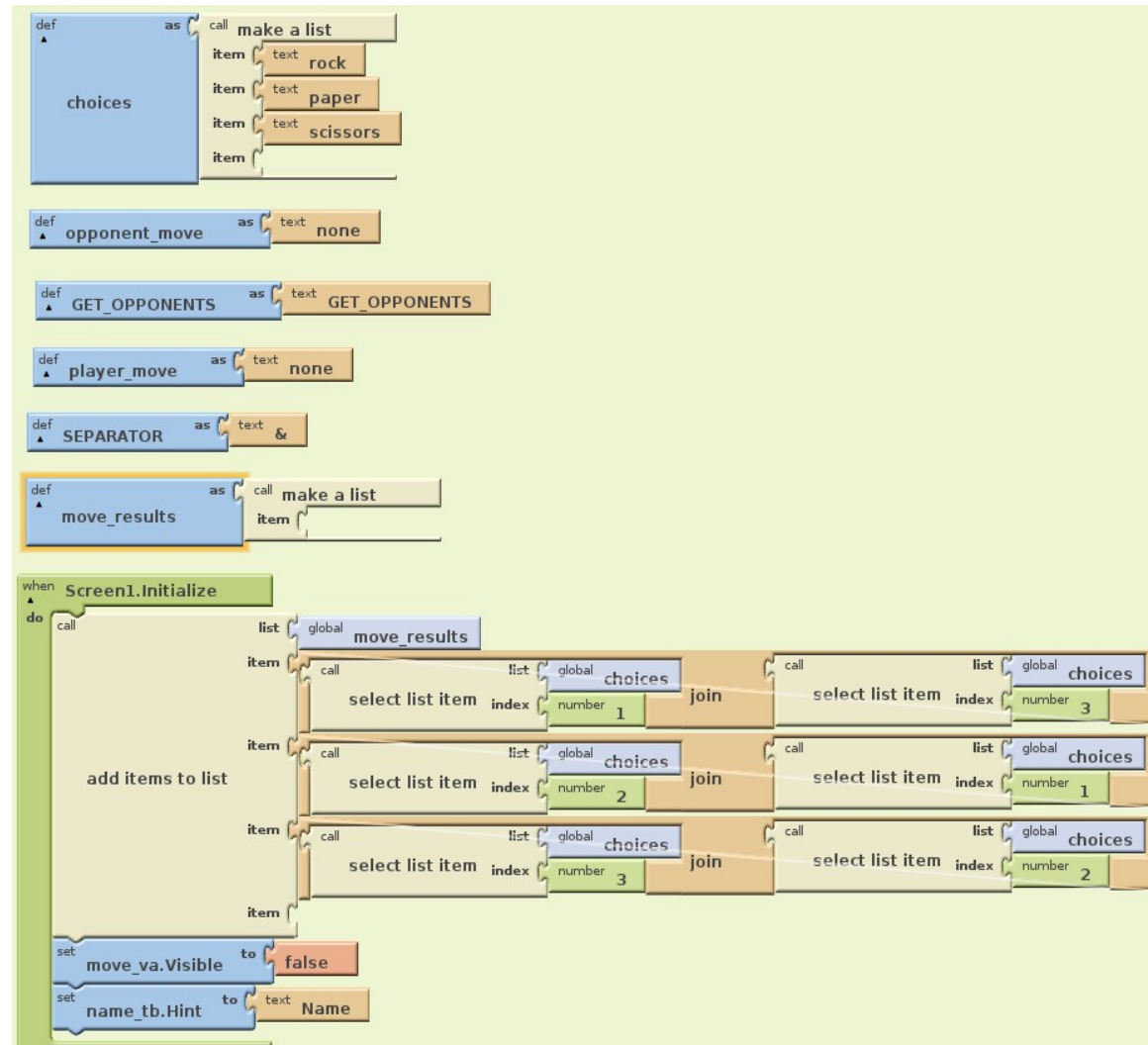
<u>Protocol</u>	<u>Layer</u>
HTTP	Application
TCP	Transport
IP	Internet
Ethernet	Link
IEEE 802.3u	Physical

Protocol specifications

- Requesting to store a player's move, `SetValue`
 - Use the player name as the key
 - Use the choice name as the value
- Requesting a list of opponents, `GetValue`
 - Use the key "GET_OPPONENTS"
 - Query returns a string value of opponent names joined by "&"
 - E.g. Request "GET_OPPONENTS" returns "Rick&Piper", i.e. Rick and Piper are possible opponents
 - Usefull blocks for handling data transfer is
 - "split", splits a string at a specific delimiter
- Requesting a player move, `GetValue`
 - Use the player name as the key
 - E.g. Request "Rick" returns the value of Rick's move

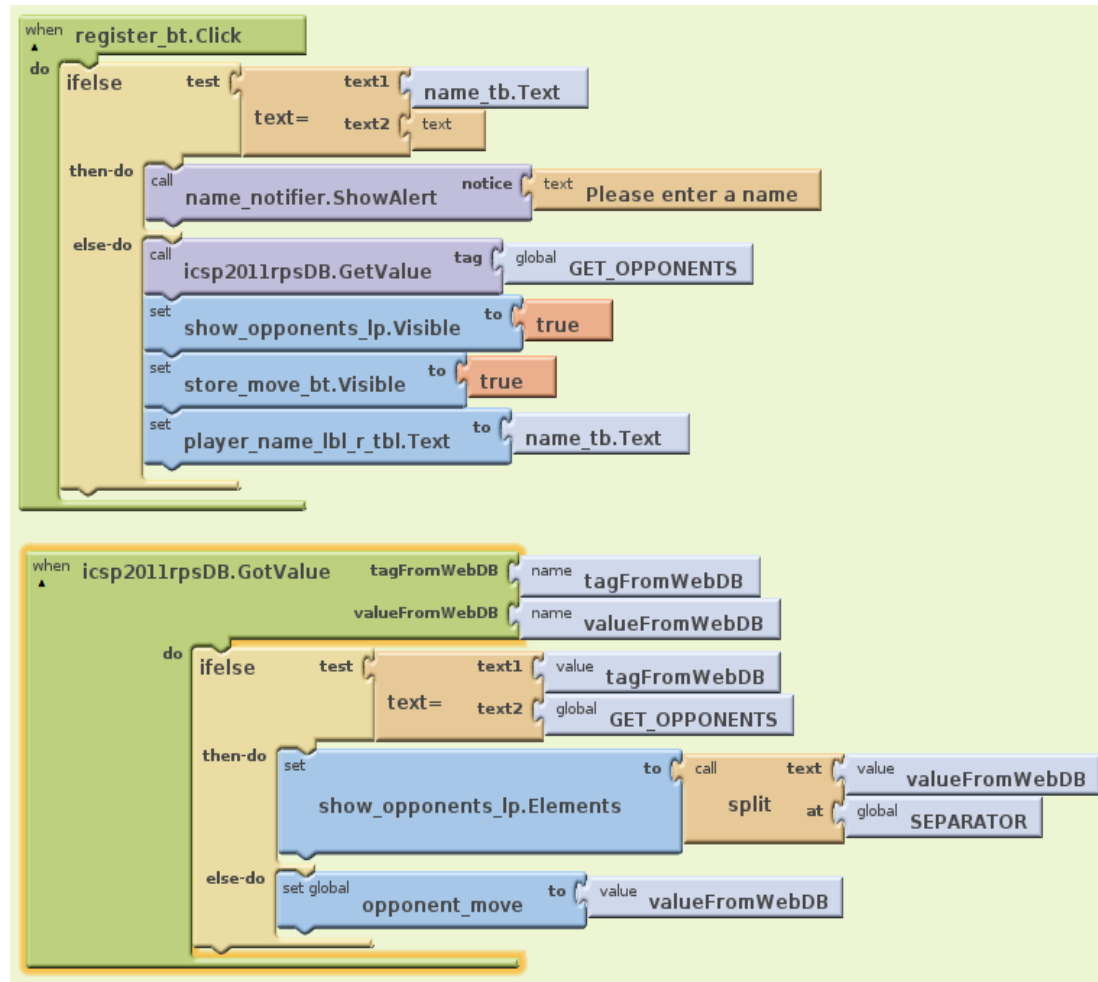
Initialize

- Define variables
- Assign values to variables
- Create the initial screen



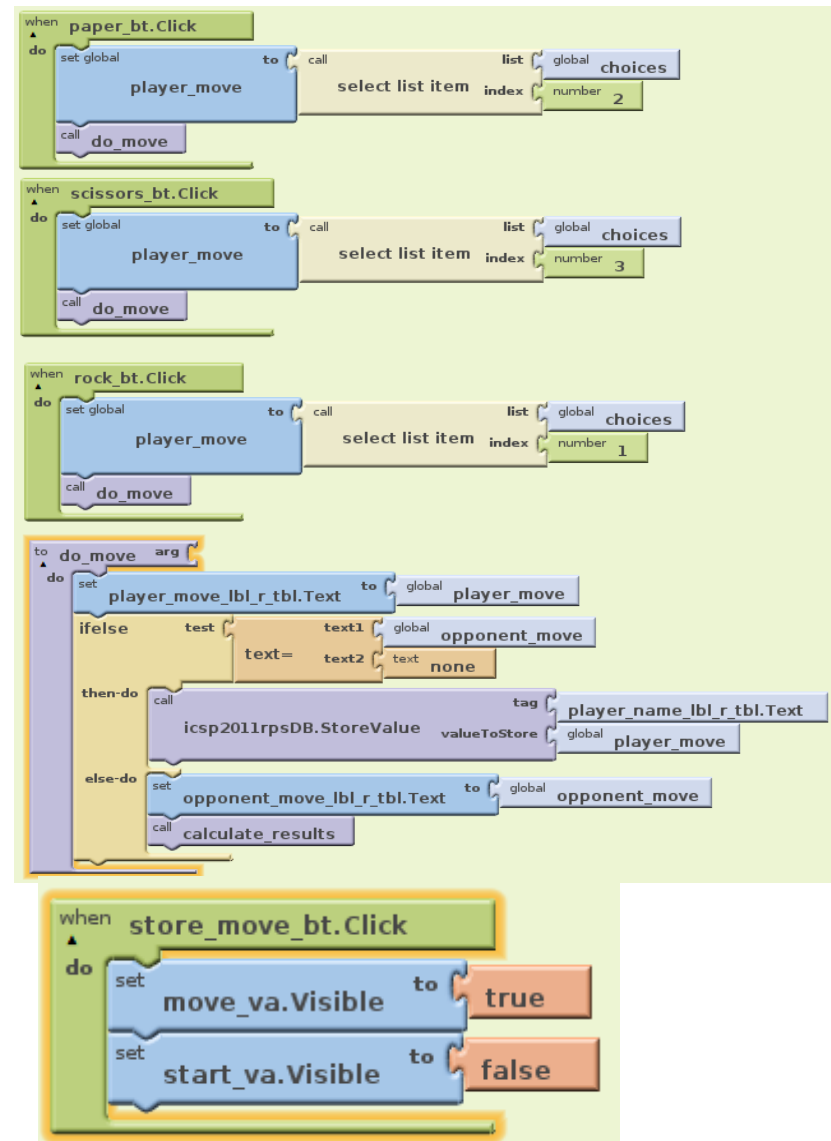
Create Register Functionality

- Register button click
- Database communication when getting a value
- Notification



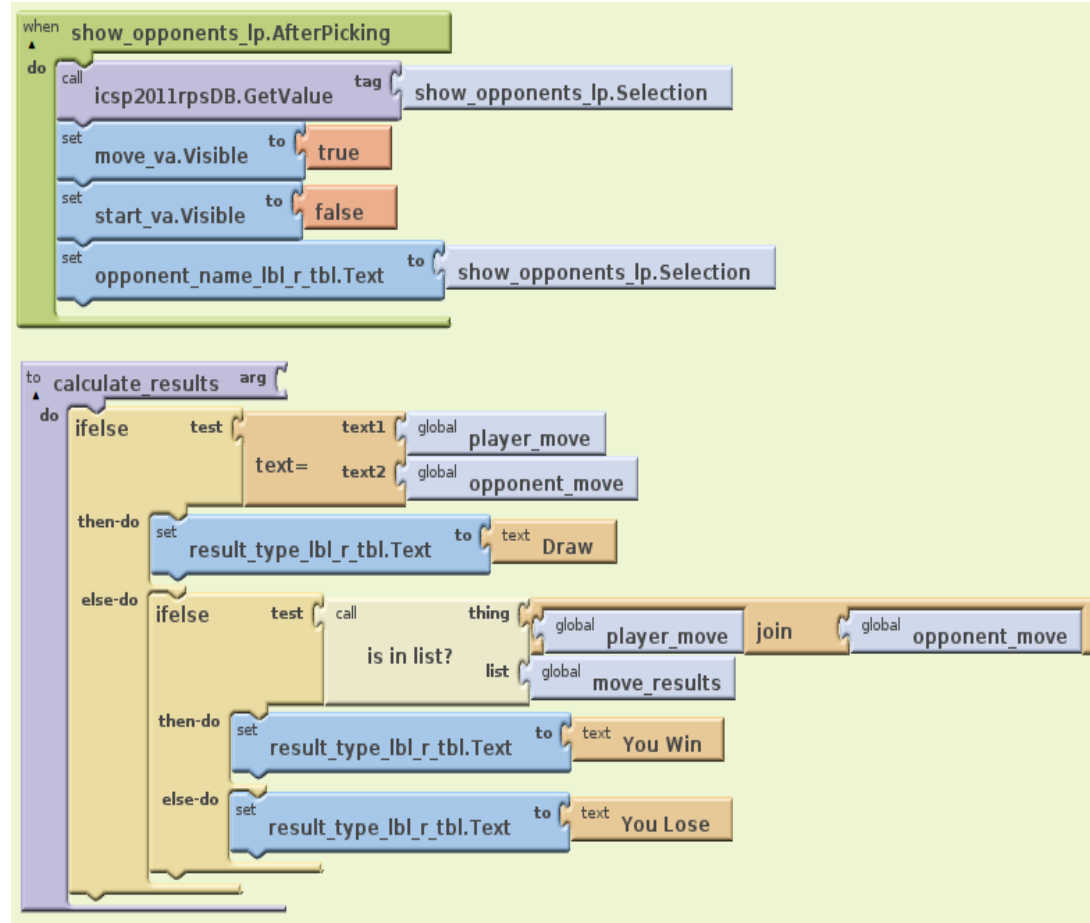
Store a Move

- Button choice behaviour
- Storing move in data base



Chose Opponents

- Methods after picking
- Calculating the result



```

when register_bt.Click
do
  ifelse test1 name_tb.Text
        test2 text=
        test3 text
  then-do
    call name_notifier.ShowAlert notice text Please enter a name
  else-do
    call icsp2011rpsDB.GetValue tag global GET_OPPONENTS
    set show_opponents_lp.Visible to true
    set store_move_bt.Visible to true
    set player_name_lbl_r_tbl.Text to name_tb.Text
  end
end

```

```

when paper_bt.Click
do
  set global player_move to call select list item list global choices
  index number 2
  call do_move
end

```

```

to calculate_results arg
do
  ifelse test1 global player_move
        test2 global opponent_move
  then-do
    set result_type_lbl_r_tbl.Text to text Draw
  else-do
    ifelse test call thing global player_move join global opponent_move
          is in list? list global move_results
    then-do
      set result_type_lbl_r_tbl.Text to text You Win
    else-do
      set result_type_lbl_r_tbl.Text to text You Lose
    end
  end
end

```

```

when scissors_bt.Click
do
  set global player_move to call select list item list global choices
  index number 3
  call do_move
end

```

```

def player_move == text none

```

```

when store_move_bt.Click
do
  set move_wa.Visible to true
  set start_wa.Visible to false
end

```

```

to do_move arg
do
  set player_move_lbl_r_tbl.Text to global player_move
  ifelse test1 global opponent_move
        test2 text none
  then-do
    call icsp2011rpsDB.StoreValue tag player_name_lbl_r_tbl.Text
    value store global player_move
  else-do
    set opponent_move_lbl_r_tbl.Text to global opponent_move
    call calculate_results
  end
end

```

```

when show_opponents_lp.AfterPicking
do
  call icsp2011rpsDB.GetValue tag show_opponents_lp.Selection
  set move_wa.Visible to true
  set start_wa.Visible to false
  set opponent_name_lbl_r_tbl.Text to show_opponents_lp.Selection
end

```

```

def choices == call make a list
item text rock
item text paper
item text scissors
end

```

```

when Screen1.Initialize
do
  list global move_results
  item call select list item list global choices index number 1 join call select list item list global choices index number 3
  item call select list item list global choices index number 2 join call select list item list global choices index number 1
  item call select list item list global choices index number 3 join call select list item list global choices index number 2
  set move_wa.Visible to false
  set name_tb.Hint to text Name
end

```

```

def opponent_move == text none

```

```

def GET_OPPONENTS == text GET_OPPONENTS

```

```

when rock_bt.Click
do
  set global player_move to call select list item list global choices
  index number 1
  call do_move
end

```

```

def SEPARATOR == text &

```

```

when icsp2011rpsDB.GetValue tagFromWebDB name tagFromWebDB
valueFromWebDB name valueFromWebDB
do
  ifelse test1 value tagFromWebDB
        test2 global GET_OPPONENTS
  then-do
    set show_opponents_lp.Elements to call text value valueFromWebDB
    split set global SEPARATOR
  else-do
    set global opponent_move to value valueFromWebDB
  end
end

```

```

def move_results == call make a list
item
end

```

Improvements of Rock, Papers or Scissors

- Disallow duplicate names
- Improve security e.g. Database injection
- Handle invalid values
- Return player to start when a move is stored
- Remove moves which has been in a game

AppInventor Notes

AppInventor Links

- <http://appinventor.googlelabs.com/learn/tutorials/index.html>
- <http://appinventor.googlelabs.com/learn/reference/index.html>
- <https://sites.google.com/site/theairepository/>
- <https://sites.google.com/site/appinventorresources/home/tutorial-topics>

Sample Apps

- General user base: <https://sites.google.com/site/theairepository/source-code>
- University of San Francisco: <http://sites.google.com/site/usfandroidmarket/>

• Debugging in AppEngine use

- “do it”
- “watch”
- AppEngine API not completely updated
- Do not use “,” when returning values with TinyWebDB
- Developing on a local server the address is “http://10.0.2.2”