MultiPlayer Rock Paper Scissors on FireBaseDB

2/26/2016
Abe Getzler

The objective of this exercise is to test FireBase DB's suitability as an AI2 vehicle to run a multiplayer game like Rock Paper Scissors (Rochambeau).

GameTypes

- RPS
    - name: “Rock Paper Scissors”
    - blurb
    - moves : [Rock, Paper, Scissors]
    - winning moves
        - Rock
            - Scissors : crushes
        - Paper
            - Rock : covers
        - Scissors

            - Paper : cuts
    - RPSSL
        - name: “[Rock Paper Scissors Spock Lizard]”
        - blurb
        - moves : [Rock, Paper, Scissors, Spock, Lizard]]
        - winning moves
            - Rock
                - Scissors : crushes
                - Lizard: crushes
            - Paper
                - Rock : covers
                - Spock: disproves
            - Scissors
                - Paper : cuts
                - Lizard : decapitates
            - Spock
                - Scissors : smashes
                - Rock : vaporizes
            - Lizard
                - Paper : eats
                - Spock : poisons

Game Types could be kept on the server, to allow for new game types being introduced from the server side. (Not implemented yet.)

Design methodology

To avoid conflicts from simultaneous updates; we will try to follow these rules on the server side

- never store summaries (counts, maxima, sums, analyses)
- never update anything in place unless you own it and no one else reads it.
- never keep lists, only subtags.
- only insert.

Also, because a FirebaseDB query with a unique tag will return as a value the JSON of everything under that tag (but not the tag), we will include a duplicate of the unique parent ID under the subtag ID, to make it accessible through the lookup-in-pairs block.

Server contents:

PlayerNames

- Player name (unique, scrub blanks and quotes)
    - ID : playerID
    - last login datetime
    - current match ID
    - challenge question (not yet implemented)
    - answer (not yet implemented)
    - newsfeed
        - matchID1 : last move YYYYMMDDHHmmss
        - matchID2 : last move YYYYMMDDHHmmss
    - matches
        - match ID YYYYMMDDHHmmss-Initiator
            - matchID : matchID (duplicated for convenience in JSON extract handling)
            - game type
            - target rounds : 3
            - target players : 2
            - players
                - player 1 ID : true
                - player 2 ID : true
                - ...
            - current round : 3
            - last move YYYYMMDDHHmmss (for cleanup)
            - rounds
                - 1
                    - player 1 ID: move
                    - player 2 ID: move
                - 2
                    - player 1 ID: move
                    - player 2 ID: move
                - …
        - ...

Players get to choose their own name, mirrored in TinyDB. Names must be registered on the FireBaseDB server to insure uniqueness. The challenge question and answer are filled in at registration time to allow the player to reclaim his Player Name into TinyDB on a new device without being rejected as a duplicate on the server side.

Players get to play multiple matches simultaneously, since opponents might be scattered world-wide.

(TODO: The newsfeed system might be unnecessary, if players monitor their current game directly in its host player subtree.)

To allow each player to have to monitor only one FirebaseDB key, the **newsfeed** subkey of each player ID has subkeys for each match that might require his attention. Other players' apps insert match IDs and last move timestamps into the news feeds of their opponents after they make moves, to trigger opponents' Data Changed events. This is a broadcast model.

Matches

- pending
        - match ID YYYYMMDDHHmmss-Initiator
            - game type (RPS/RPSSL)
            - target rounds : 3
            - target players : 2
            - players
                - player 1 ID : true
                - ...
            - ...
- running
    - match ID YYYYMMDDHHmmss-Initiator : true
    - match ID YYYYMMDDHHmmss-Initiator : true
    - ...

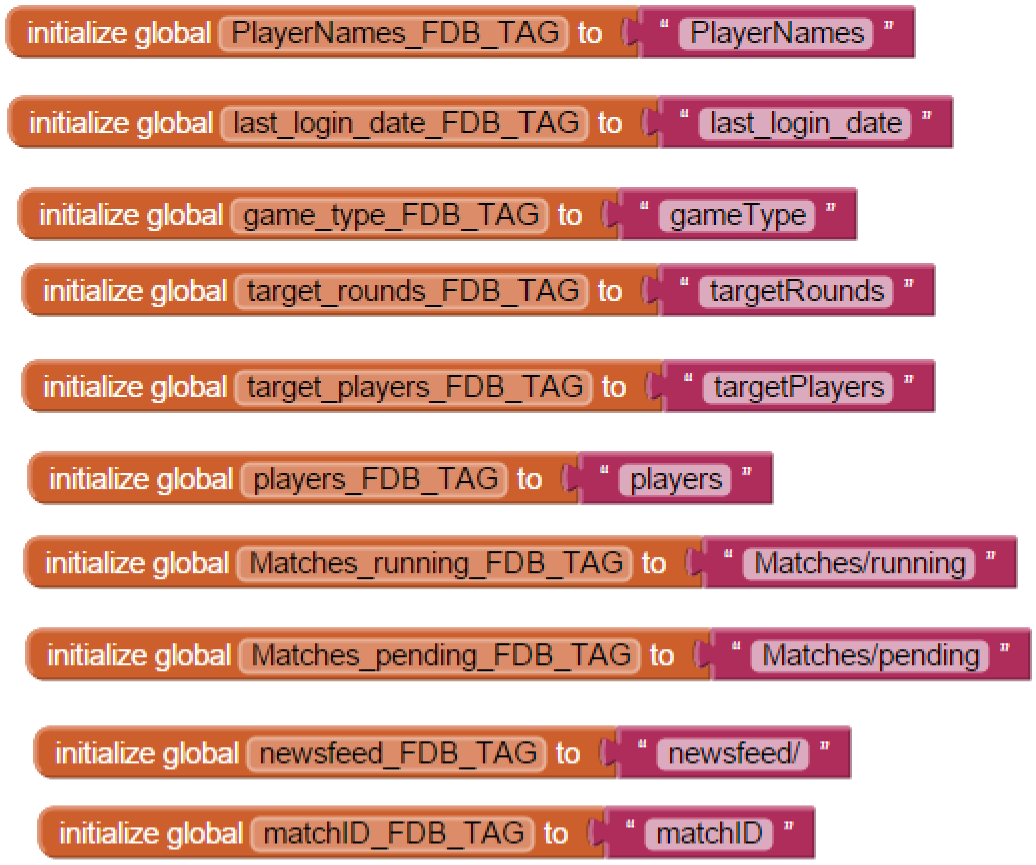
There are two legs to the Matches branch: pending and running, of interest to people who want to join or watch a match, respectively.

Match IDs are designed to insure uniqueness (no guids are available), for chronological cleanup, and for possible filtering by initiator player ID. Pending matches do not yet have the required minimum number of players to start. Once a pending match has enough players, the app of the last player to join transfers it to the Running section and removes it from the Pending section.
Once a match completes, the last player to move removes it from the running branch.

The Initiator of a match stores the match information under his PlayerID, and he and the other Players insert their moves into that match tree as the game progresses, and monitor that subtree if they are playing or watching that match.

FireBaseDB Tags and Subtags

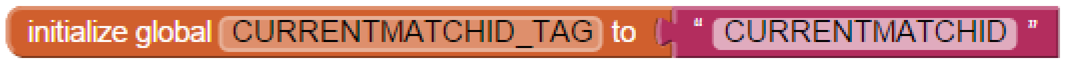
To allow FirebaseDB to return JSON strings for tags with subtags (/ separator), we have to exclude spaces from our Firebase tags. Here I have used underscores and CamelCase to highlight the words in my tags.



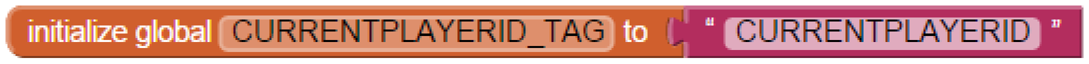
```
initialize global PlayerNames_FDB_TAG to " PlayerNames "
initialize global last_login_date_FDB_TAG to " last_login_date "
initialize global game_type_FDB_TAG to " gameType "
initialize global target_rounds_FDB_TAG to " targetRounds "
initialize global target_players_FDB_TAG to " targetPlayers "
initialize global players_FDB_TAG to " players "
initialize global Matches_running_FDB_TAG to " Matches/running "
initialize global Matches_pending_FDB_TAG to " Matches/pending "
initialize global newsfeed_FDB_TAG to " newsfeed/ "
initialize global matchID_FDB_TAG to " matchID "
```

TinyDB tags

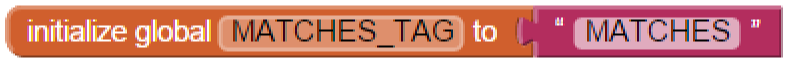
- CURRENTPLAYERID
- CURRENTMATCHID
- MATCHES - a list of this owner’s matchIDs, hosted or not



```
initialize global CURRENTMATCHID_TAG to " CURRENTMATCHID "
```



```
initialize global CURRENTPLAYERID_TAG to " CURRENTPLAYERID "
```



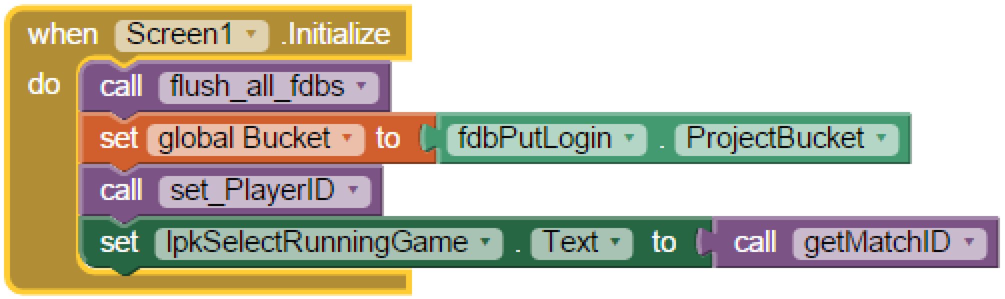
```
initialize global MATCHES_TAG to " MATCHES "
```

App Dialogs

Initialization

globals

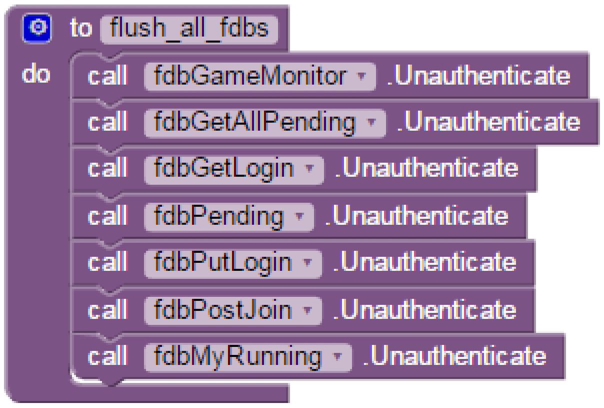
Screen1.Initialize



```
when Screen1 .Initialize
do  call flush_all_fdbs
    set global Bucket to fdbPutLogin . ProjectBucket
    call set_PlayerID
    set lpkSelectRunningGame . Text to call getMatchID
```

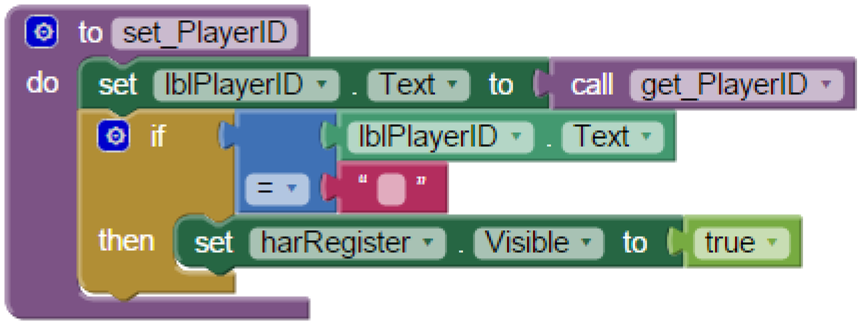
At Initialization time, we want to show the current PlayerID from TinyDB. If there isn't one, show the Registration fields. We need to save the base FirebaseDB project bucket so that we can extend it later for newsfeed monitoring.
References: set_PlayerID, flush_all_fdbs

flush_all_fdbs



```
to flush_all_fdbs
do  call fdbGameMonitor .Unauthenticate
    call fdbGetAllPending .Unauthenticate
    call fdbGetLogin .Unauthenticate
    call fdbPending .Unauthenticate
    call fdbPutLogin .Unauthenticate
    call fdbPostJoin .Unauthenticate
    call fdbMyRunning .Unauthenticate
```

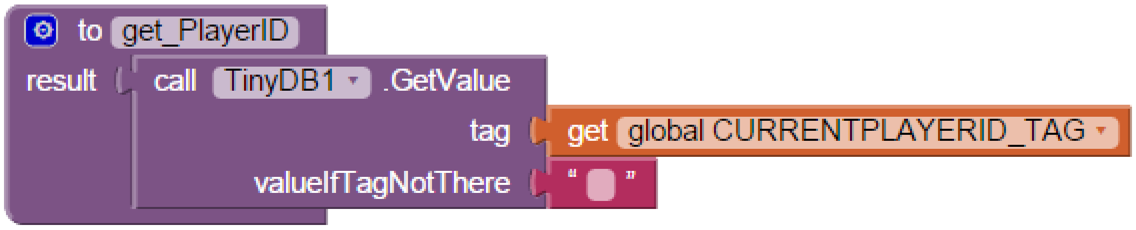
set_PlayerID



```
to set_PlayerID
do  set lblPlayerID . Text to  call get_PlayerID
    if    lblPlayerID . Text
          =  " "
    then  set harRegister . Visible to  true
```

This user's PlayerID is kept in TinyDB. If there is no PlayerID available, expose the Register Horizontal Arrangement.

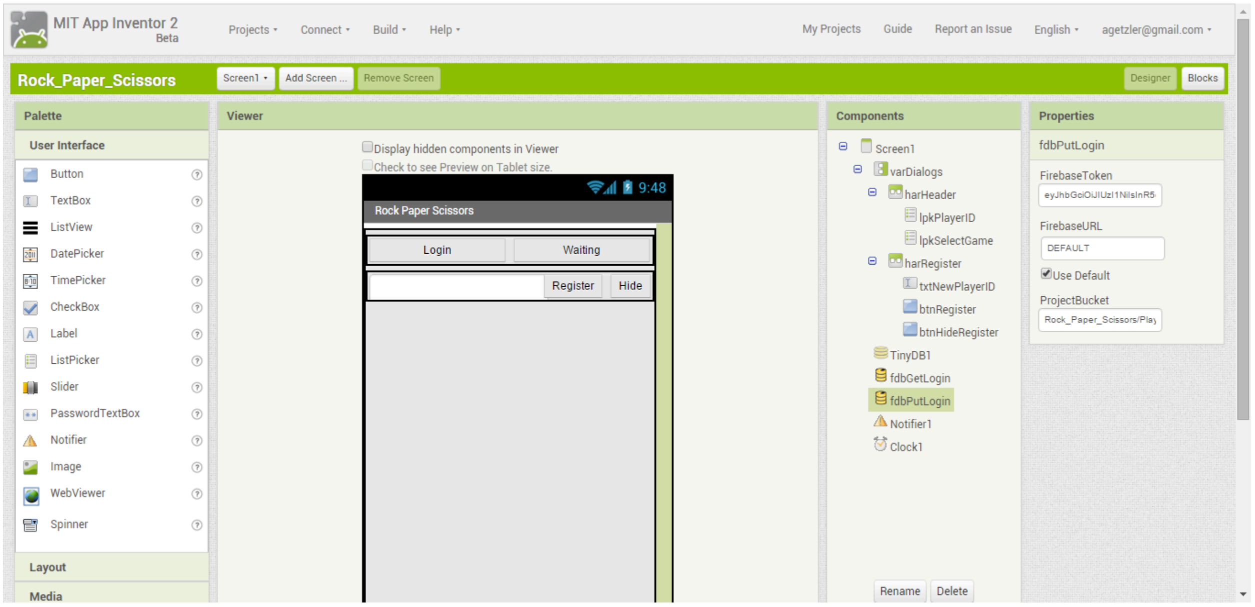
get_PlayerID



```
to get_PlayerID
result  call TinyDB1 .GetValue
                     tag  get global CURRENTPLAYERID_TAG
         valueIfTagNotThere  " "
```

Status header: current playerID, newsfeed, selected matchID,

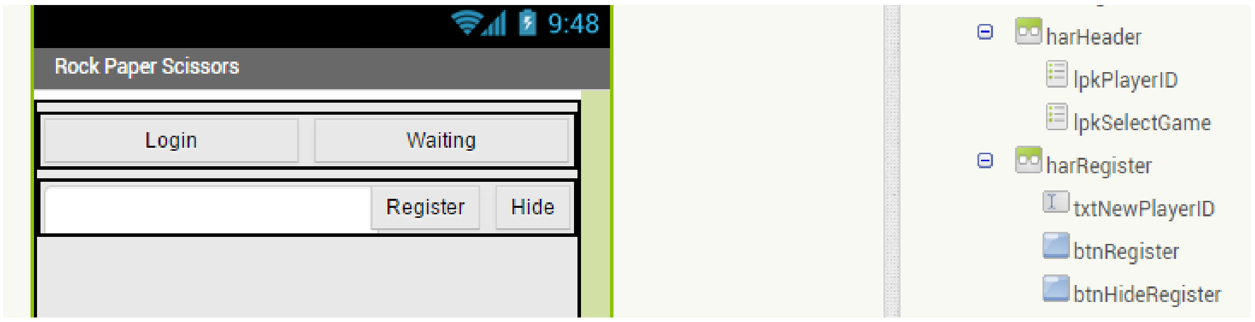
Player Login Designer



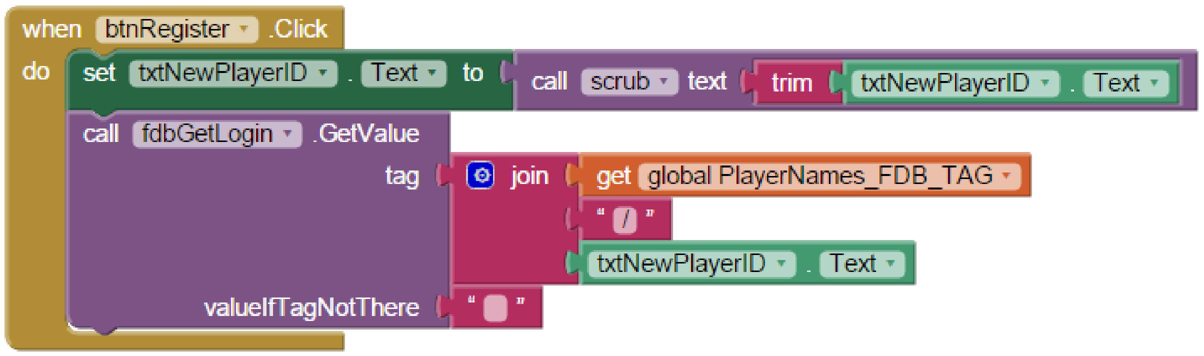
FirebaseToken: eyJhbGciOiJIUzI1NiIsInR5
FirebaseURL: DEFAULT
☑ Use Default
ProjectBucket: Rock_Paper_Scissors/Play

Login / Register

Designer Dialog for Registering

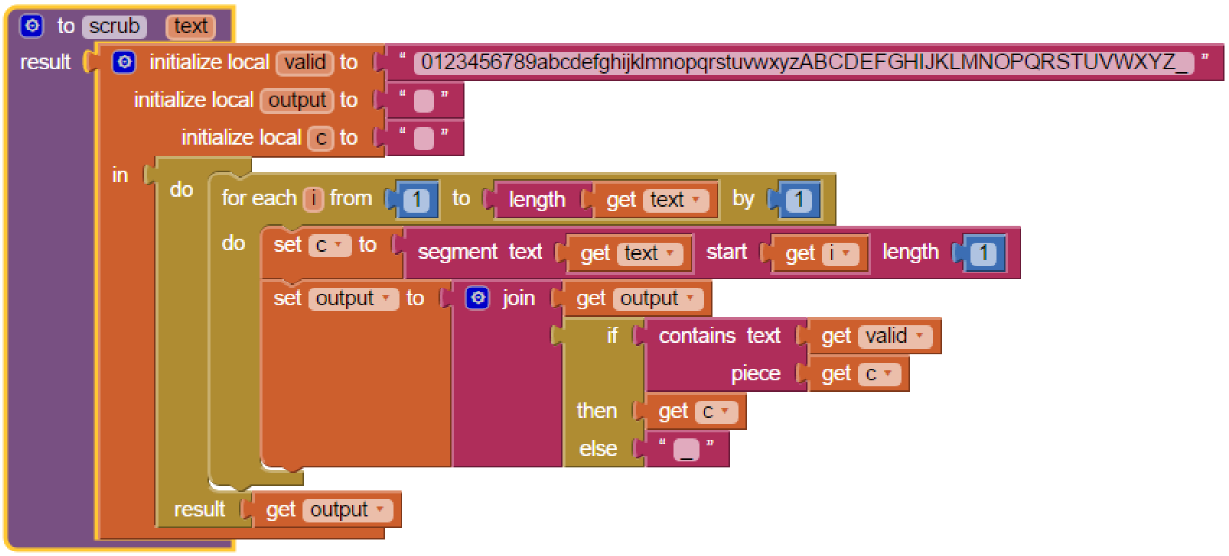


btnRegister.Click



```
when btnRegister .Click
do  set txtNewPlayerID . Text to  call scrub text  trim  txtNewPlayerID . Text
    call fdbGetLogin .GetValue
                    tag  join  get global PlayerNames_FDB_TAG
                               " / "
                               txtNewPlayerID . Text
        valueIfTagNotThere  " "
```

scrub procedure

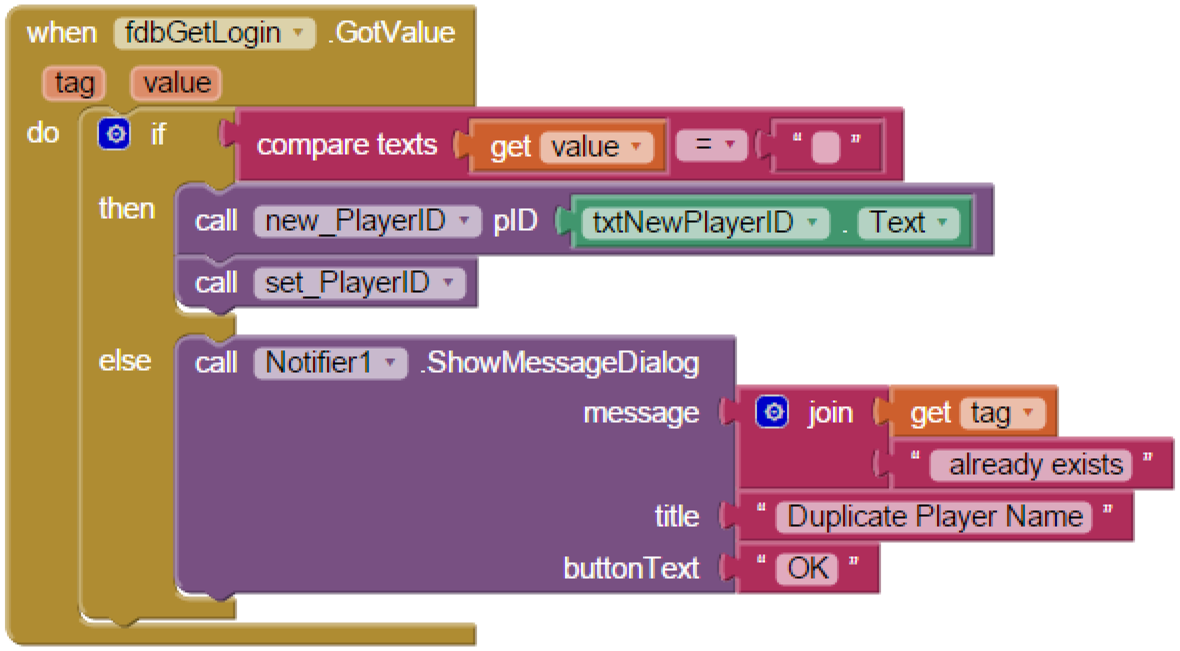


```
to scrub text
result  initialize local valid to "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_"
        initialize local output to " "
        initialize local c to " "
        in  do  for each i from 1 to length get text by 1
                do  set c to segment text get text start get i length 1
                    set output to  join  get output
                                         if   contains text get valid
                                                       piece get c
                                         then get c
                                         else "_"
            result  get output
```

Anything that isn't an upper or lower case letter or number is replaced with '_'.

Registering a new PlayerID is a two phase process. All PlayerIDs are trimmed of trailing blanks, and retrieved from the PlayerNames branch of FirebaseDB.

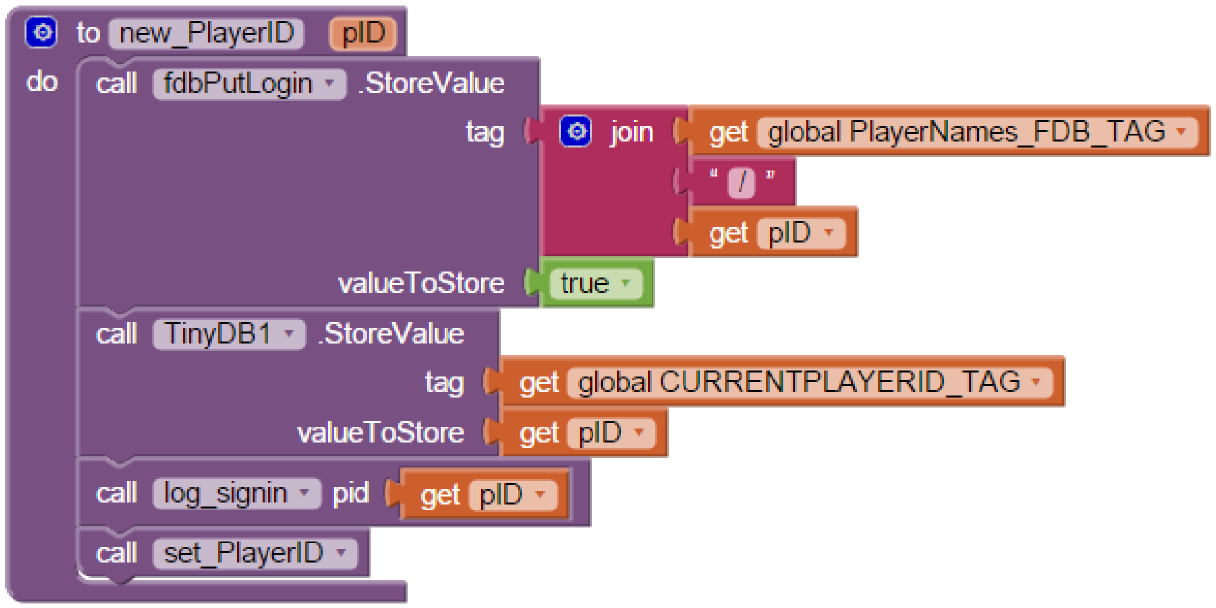
fdbGetLogin.GotValue



```
when fdbGetLogin .GotValue
  tag  value
do  if  compare texts  get value  =  " "
    then  call new_PlayerID  pID  txtNewPlayerID . Text
          call set_PlayerID
    else  call Notifier1 .ShowMessageDialog
                               message  join  get tag
                                              " already exists "
                                 title  " Duplicate Player Name "
                            buttonText  " OK "
```

If the returned PlayerID from FireBaseDB is blank, it's a new ID, so we proceed to add it using procedure new_PlayerID. Otherwise we alert the user.

new_PlayerID



```
to new_PlayerID pID
do  call fdbPutLogin .StoreValue
                   tag    join  get global PlayerNames_FDB_TAG
                                " / "
                                get pID
          valueToStore   true
    call TinyDB1 .StoreValue
                   tag   get global CURRENTPLAYERID_TAG
          valueToStore   get pID
    call log_signin pid  get pID
    call set_PlayerID
```

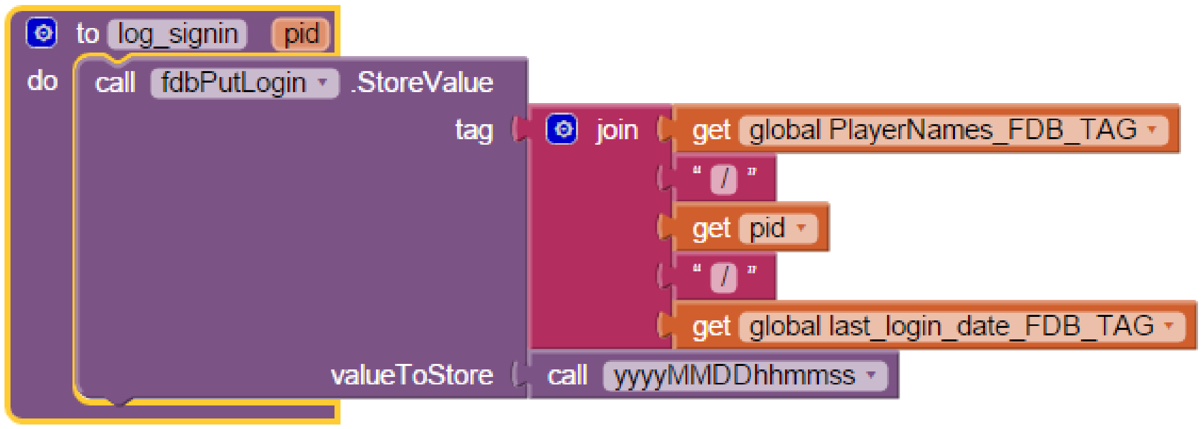
Player IDs are stored as subtags under a constant FireBaseDB tag, the global PlayerNames_FDB_TAG. The “/” starts a new JSON subtree . The “true” value is a place holder, to be replaced by subfields later on.

The new PlayerID is taken as the current one, in TinyDB and through procedure log_signin.

References: log_signin, set_PlayerID.

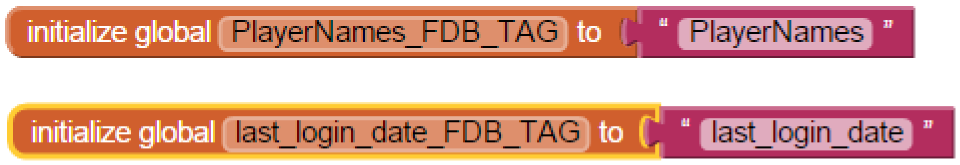
log_signin

A last_login_date datetime value is kept to allow tracking and cleanup of dead PlayerIDs.



```
to log_signin pid
do  call fdbPutLogin .StoreValue
                        tag  join  get global PlayerNames_FDB_TAG
                                   " / "
                                   get pid
                                   " / "
                                   get global last_login_date_FDB_TAG
               valueToStore  call yyyyMMDDhhmmss
```

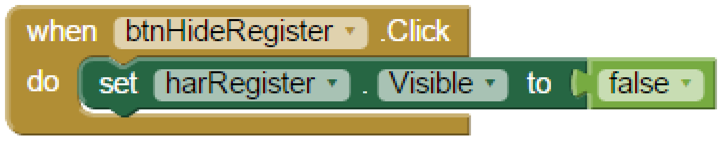
login date tag values



```
initialize global PlayerNames_FDB_TAG to " PlayerNames "
initialize global last_login_date_FDB_TAG to " last_login_date "
```

All tags, both TinyDB and FireBaseDB, are accessed through global variables, to avoid typos and to take advantage of typeblocking at block edit time.

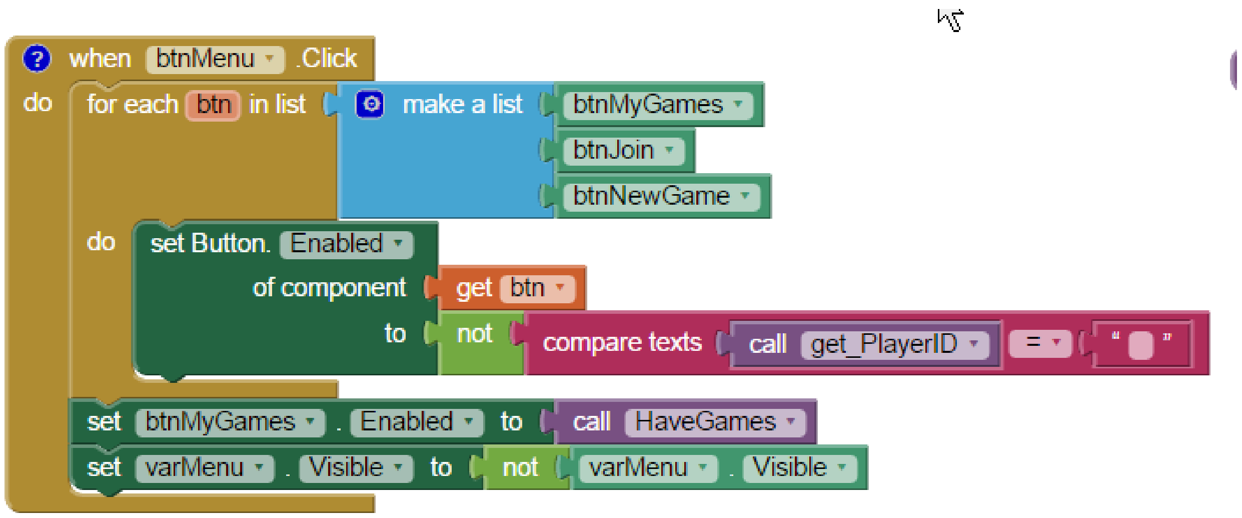
btnHideRegister



```
when btnHideRegister .Click
do  set harRegister . Visible to false
```

A Hide button in the Register Arrangement allows the user to hide it until he requests a new PlayerID.

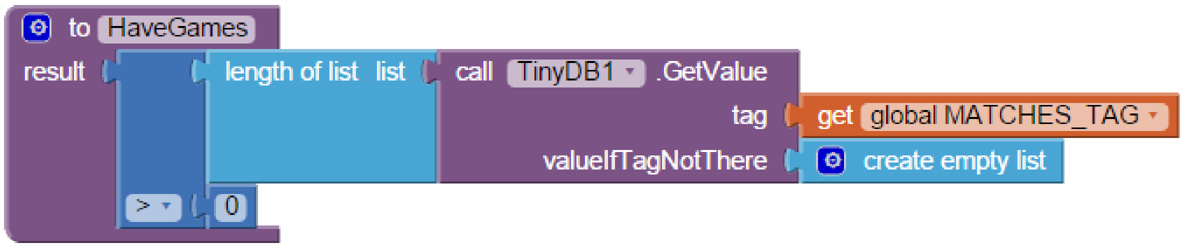
Menu button



```
when btnMenu .Click
do  for each btn in list  make a list  btnMyGames
                                       btnJoin
                                       btnNewGame
    do  set Button. Enabled
                 of component  get btn
                           to  not  compare texts  call get_PlayerID  =  " "
    set btnMyGames . Enabled to  call HaveGames
    set varMenu . Visible to  not  varMenu . Visible
```

If the user hasn't yet picked a Player Name, he can't ask for his running games, join a pending game, or start a new game. The Menu button exposes a Vertical Arrangement with more action buttons. References: get_PlayerID, HaveRunningGames.

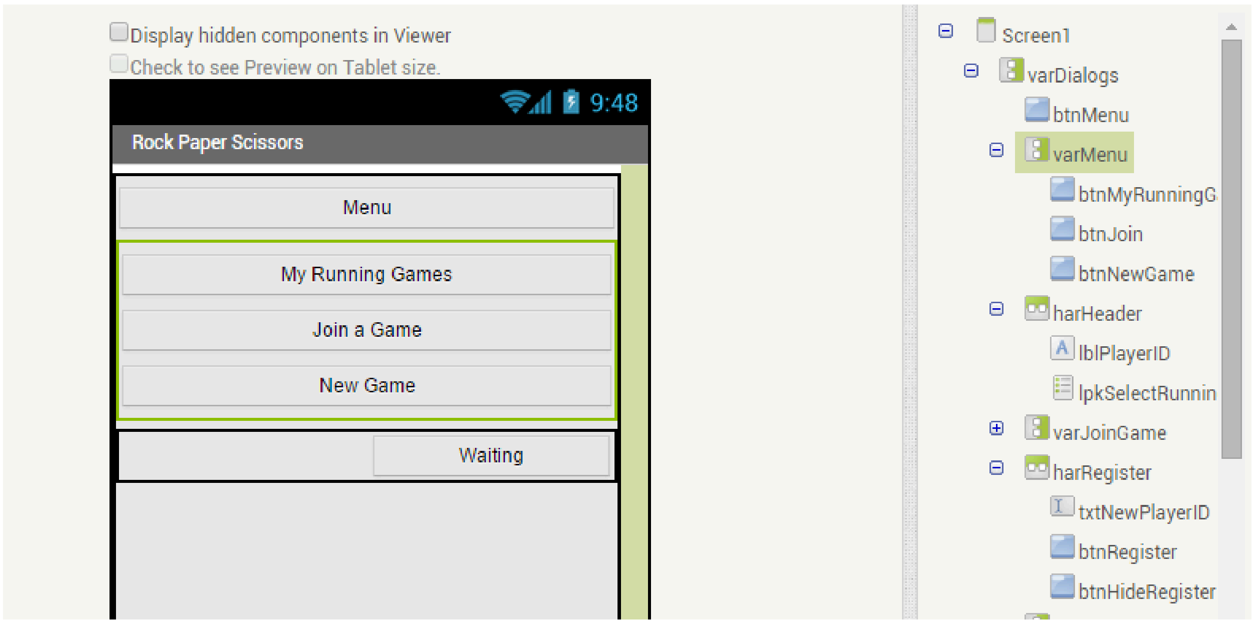
HaveGames



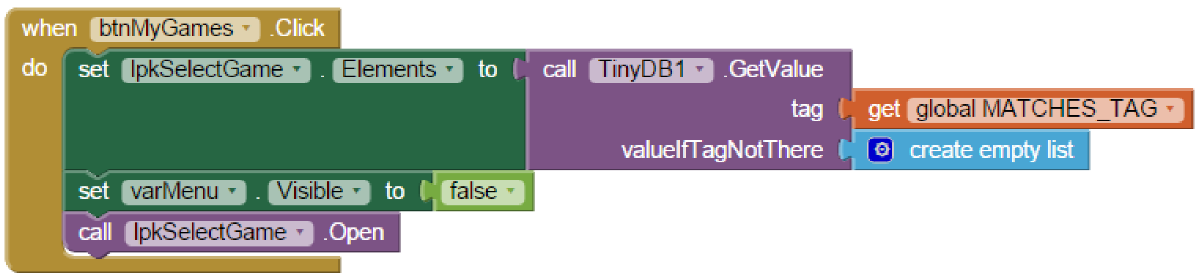
```
to HaveGames
result  length of list  list  call TinyDB1 .GetValue
                                         tag  get global MATCHES_TAG
                              valueIfTagNotThere  create empty list
        >  0
```

The My Running Games button is disabled if the player has no running games. Since a list is expected, we return a default value of an empty list.

Designer Menu arrangement

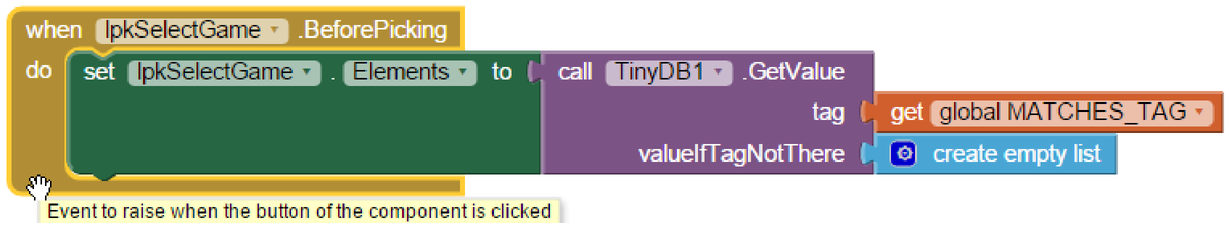


btnMyGames.Click



```
when btnMyGames .Click
do  set lpkSelectGame . Elements to  call TinyDB1 .GetValue
                                                      tag  get global MATCHES_TAG
                                         valueIfTagNotThere  create empty list
    set varMenu . Visible to  false
    call lpkSelectGame .Open
```

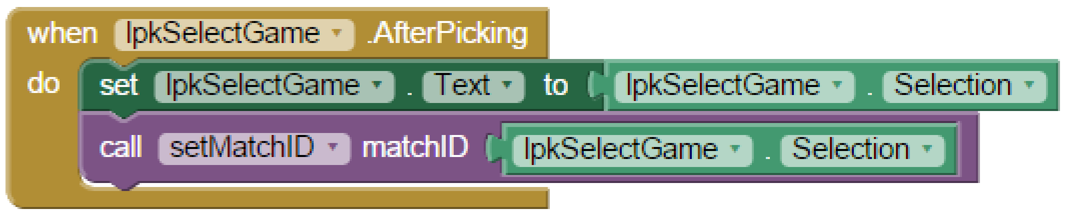
lpkSelectGame.BeforePicking



```
when lpkSelectGame .BeforePicking
do  set lpkSelectGame . Elements to  call TinyDB1 .GetValue
                                                      tag  get global MATCHES_TAG
                                         valueIfTagNotThere  create empty list
```

Event to raise when the button of the component is clicked

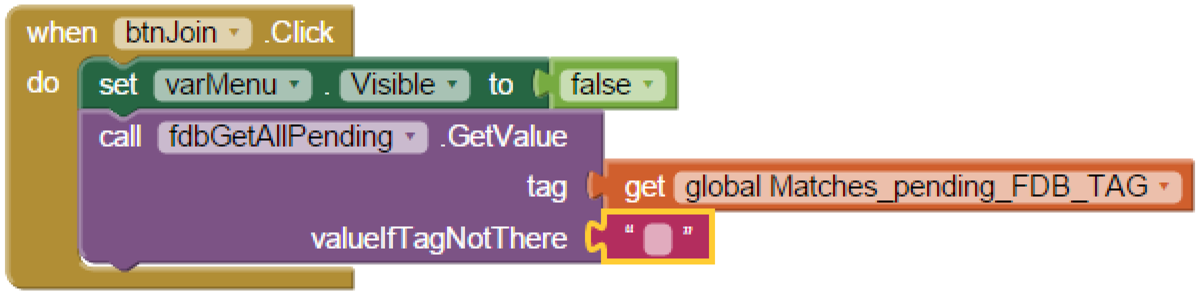
lpkSelectGame.AfterPicking



```
when lpkSelectGame .AfterPicking
do  set lpkSelectGame . Text to  lpkSelectGame . Selection
    call setMatchID matchID  lpkSelectGame . Selection
```

Reference: setMatchID

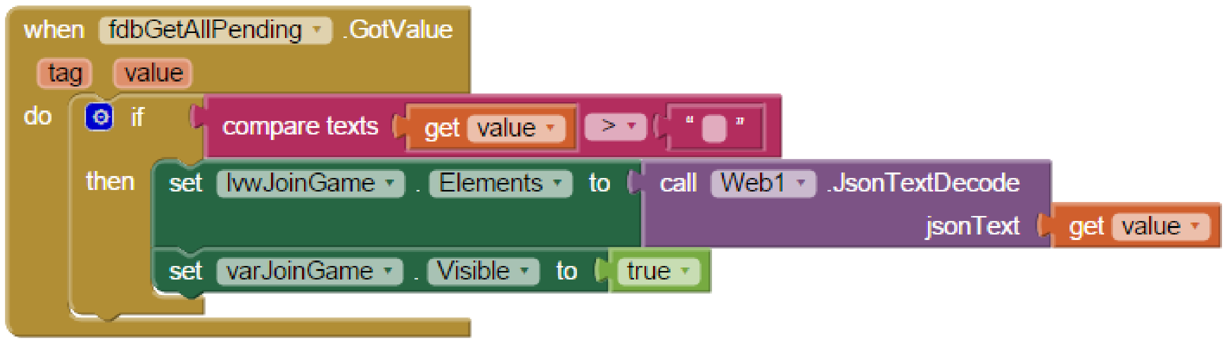
btnJoin.Click



```
when btnJoin .Click
do  set varMenu . Visible to false
    call fdbGetAllPending .GetValue
                          tag get global Matches_pending_FDB_TAG
            valueIfTagNotThere " "
```

The Menu Join button does not itself do a join. It prompts FirebaseDB for a list of pending matches that he can select from and join.

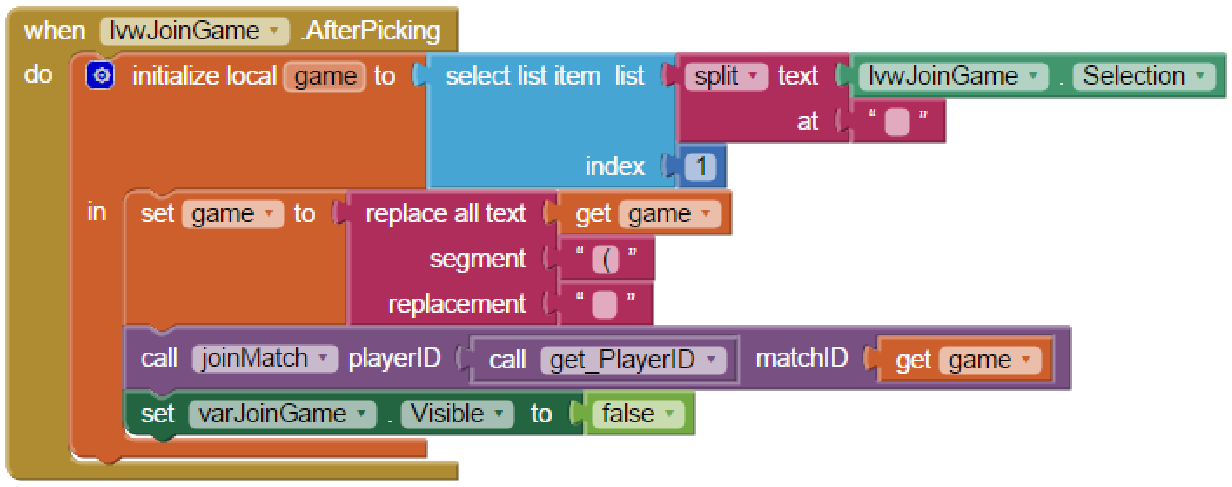
fdbGetAllPending.GotValue



```
when fdbGetAllPending .GotValue
 tag  value
do  if  compare texts get value > " "
    then set lwJoinGame . Elements to call Web1 .JsonTextDecode
                                               jsonText get value
         set varJoinGame . Visible to true
```

When Firebase comes back with the JSON tree of all pending games, we decode the JSON and load it into a ListView, and make it visible for selection. The Web1.JSONTextDecode block is explained at the MIT web site, http://ai2.appinventor.mit.edu/reference/components/connectivity.html#Web and also see this link for how to navigate a tree: http://ai2.appinventor.mit.edu/reference/other/xml.html.

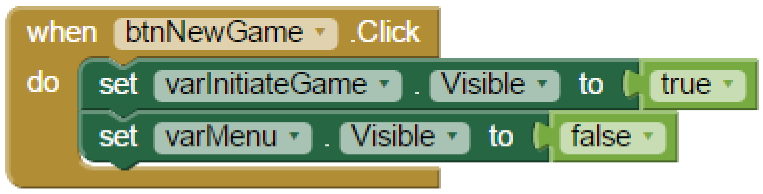
lvwJoinGame.AfterPicking



```
when lvwJoinGame .AfterPicking
do  initialize local game to  select list item list  split text  lvwJoinGame . Selection
                                                            at  " "
                                                   index  1
    in  set game to  replace all text  get game
                     segment  " ( "
                     replacement  " "
        call joinMatch playerID  call get_PlayerID  matchID  get game
        set varJoinGame . Visible to  false
```

A ListView Selection is forced to be text, so it has to be split and stripped to extract the gameID. References: joinMatch, get_PlayerID.

btnNewGame.Click



```
when btnNewGame .Click
do  set varInitiateGame . Visible to  true
    set varMenu . Visible to  false
```

Running Matches awaiting your move

- view completed rounds
- make your move for the current round

Running matches awaiting other player moves

- refresh button

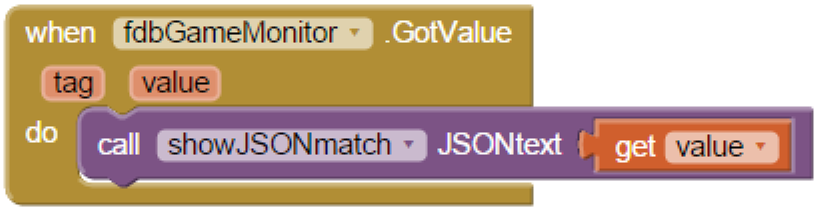
Join a pending match

- view pending matches by type
- select a match

    - join the match

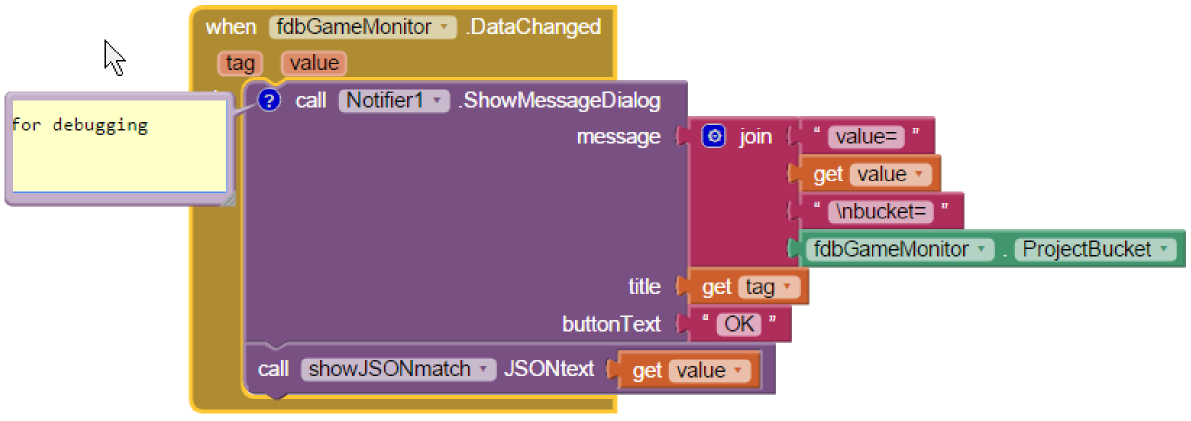
Monitor a match

fdbGameMonitor.GotValue



```
when fdbGameMonitor .GotValue
  tag  value
do  call showJSONmatch JSONtext  get value
```

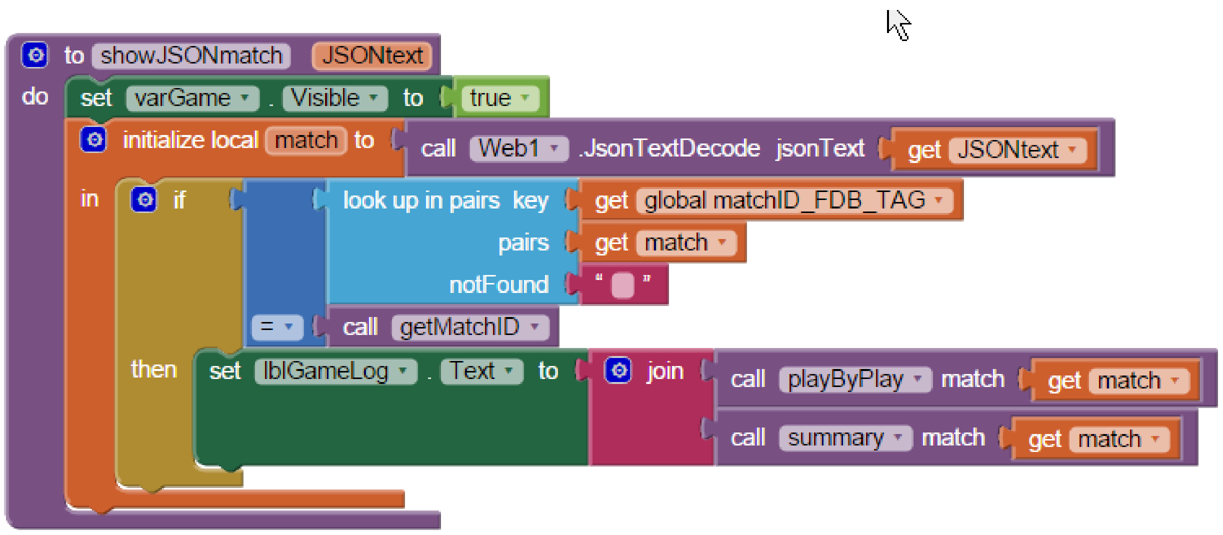
fdbGameMonitor.DataChanged



```
for debugging

when fdbGameMonitor .DataChanged
  tag  value
do  call Notifier1 .ShowMessageDialog
                  message  join  " value= "
                                 get value
                                 " \nbucket= "
                                 fdbGameMonitor . ProjectBucket
                    title  get tag
               buttonText  " OK "
    call showJSONmatch JSONtext  get value
```

showJSONmatch

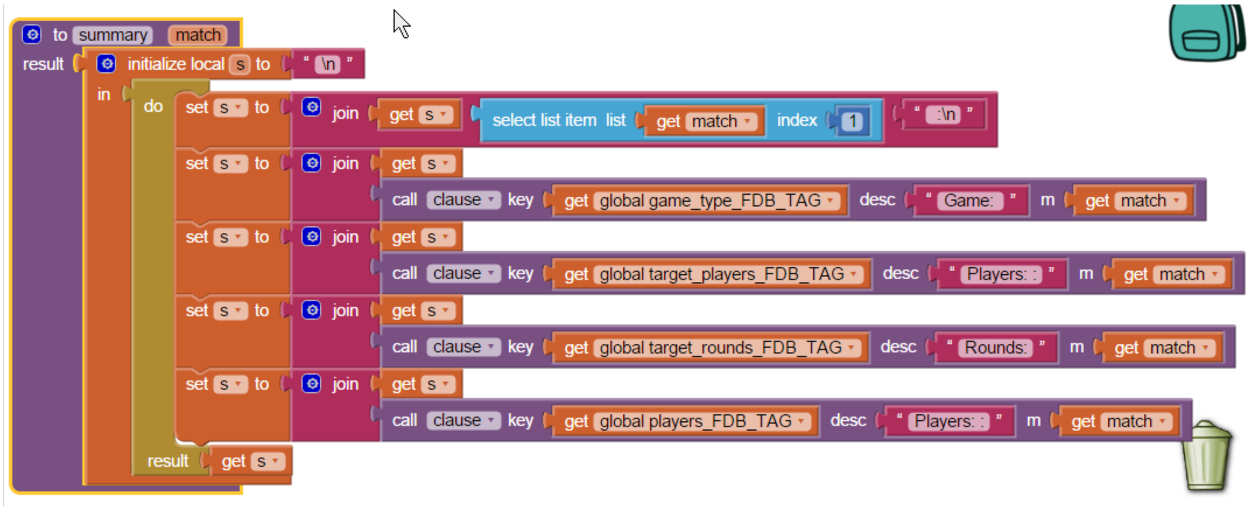


```
to showJSONmatch JSONtext
do  set varGame . Visible to  true
    initialize local match to  call Web1 .JsonTextDecode jsonText  get JSONtext
    in  if    look up in pairs key  get global matchID_FDB_TAG
                             pairs  get match
                          notFound  " "
              =
              call getMatchID
        then  set lblGameLog . Text to  join  call playByPlay match  get match
                                              call summary match  get match
```

Called by: fdbGameMonitor.GotValue, fdbGameMonitor.DataChanged.

References: summary, playByPlay, getMatchID.

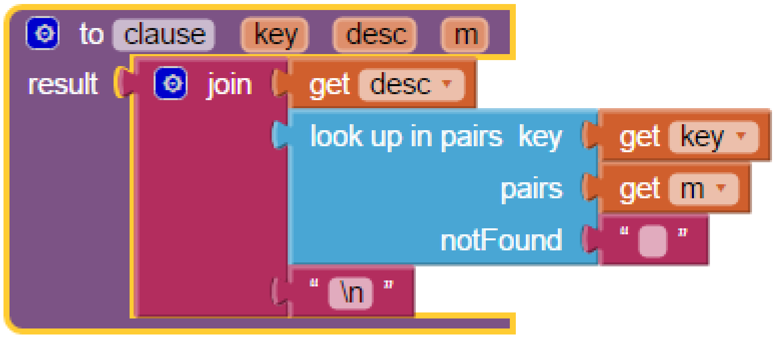
summary



```
to summary match
result  initialize local s to " \n "
        in  do  set s to join  get s
                                select list item  list  get match
                                                  index 1
                                " :\n "
                set s to join  get s
                               call clause  key  get global game_type_FDB_TAG  desc " Game: "  m  get match
                set s to join  get s
                               call clause  key  get global target_players_FDB_TAG  desc " Players: : "  m  get match
                set s to join  get s
                               call clause  key  get global target_rounds_FDB_TAG  desc " Rounds: "  m  get match
                set s to join  get s
                               call clause  key  get global players_FDB_TAG  desc " Players: : "  m  get match
            result  get s
```

Called by: showJSONmatch.
References: clause.

clause

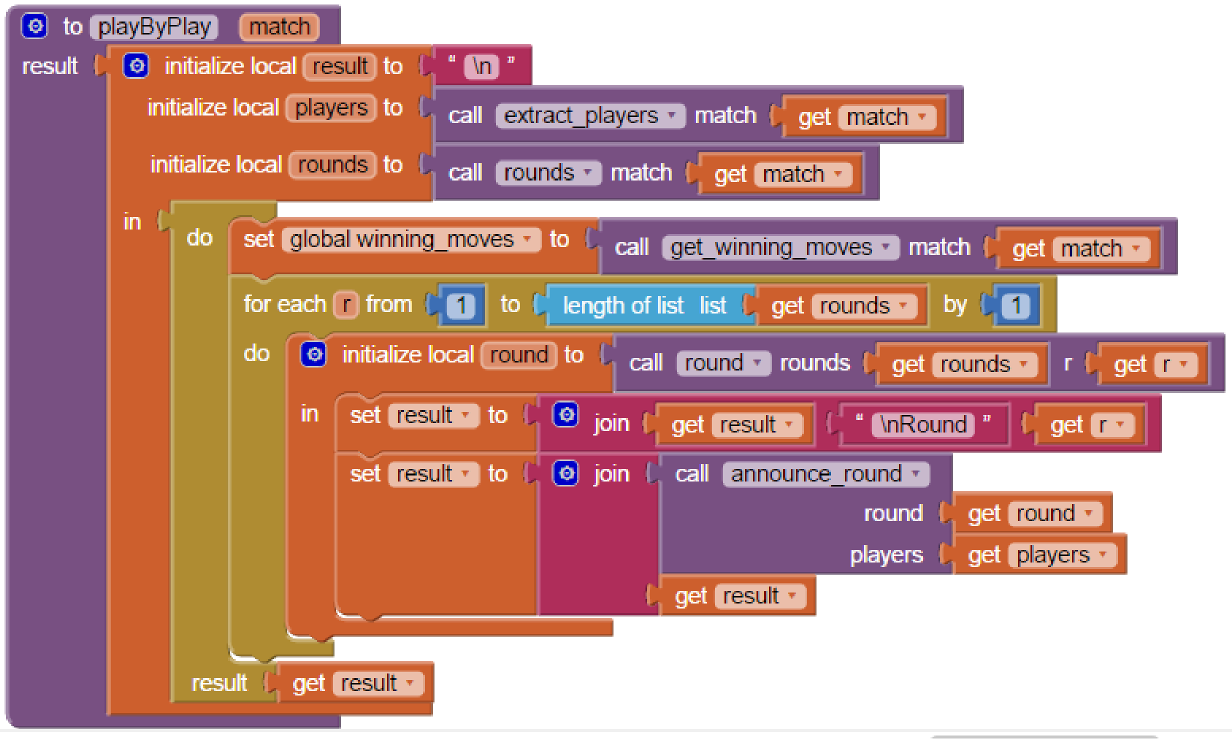


```
to clause key desc m
result  join  get desc
              look up in pairs  key       get key
                                pairs     get m
                                notFound  "  "
              " \n "
```

A game summary consists of series of clauses, each with its own subkey of a match tree and a description.

Called by: summary.

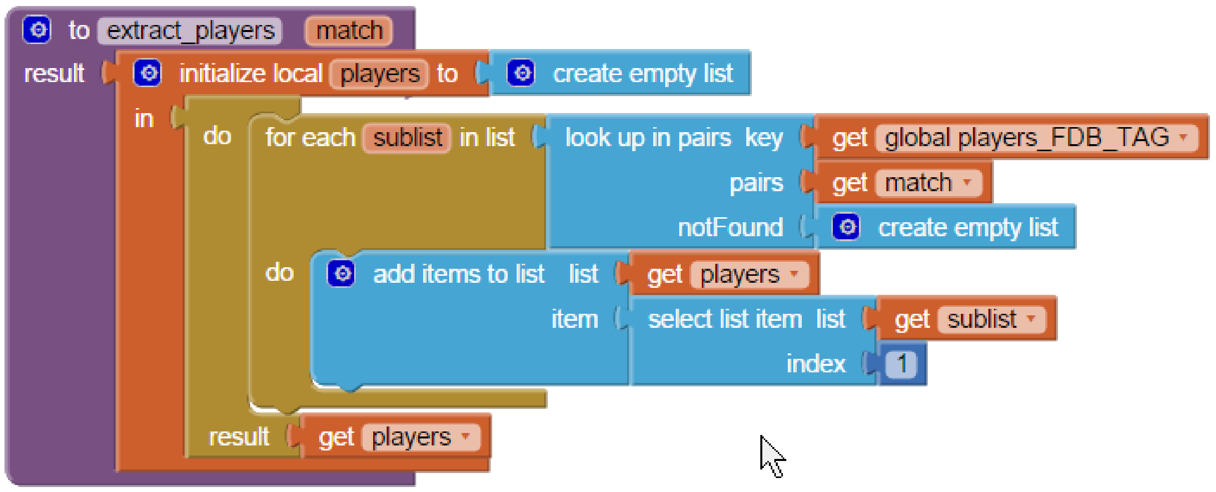
playByPlay



```
to playByPlay match
result  initialize local result to " \n "
        initialize local players to  call extract_players match  get match
        initialize local rounds to  call rounds match  get match
        in  do  set global winning_moves to  call get_winning_moves match  get match
                for each r from 1 to length of list list  get rounds  by 1
                do  initialize local round to  call round rounds  get rounds  r  get r
                    in  set result to  join  get result
                                             " \nRound "
                                             get r
                        set result to  join  call announce_round
                                                 round  get round
                                                 players  get players
                                             get result
            result  get result
```

Called by: showJSONmatch.
References: extract_players,

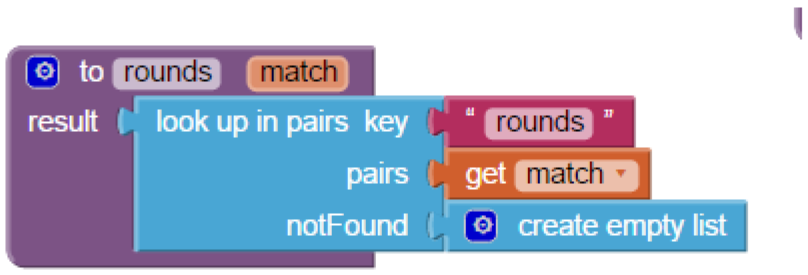
extract_players



```
to extract_players match
result  initialize local players to  create empty list
        in  do  for each sublist in list  look up in pairs  key  get global players_FDB_TAG
                                                            pairs  get match
                                                            notFound  create empty list
                do  add items to list  list  get players
                                       item  select list item  list  get sublist
                                                               index  1
            result  get players
```

Called by: playByPlay.

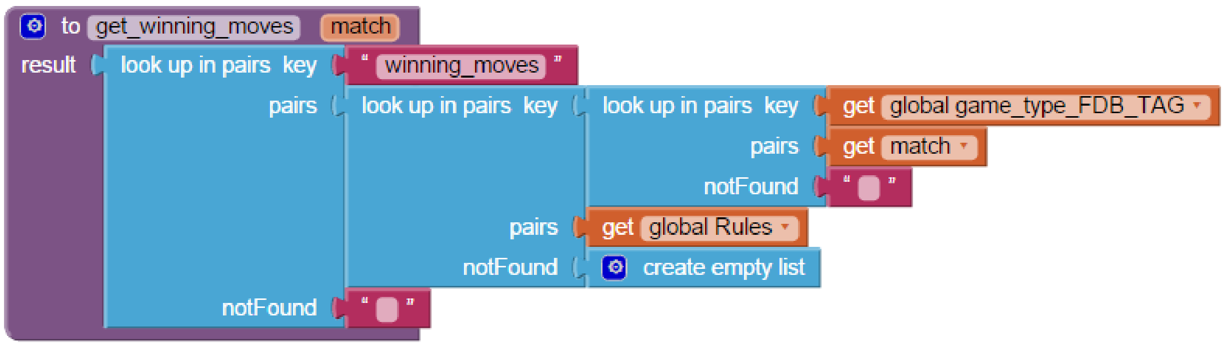
rounds



```
to rounds match
result  look up in pairs  key      " rounds "
                          pairs    get match
                          notFound create empty list
```

Called by: playByPlay

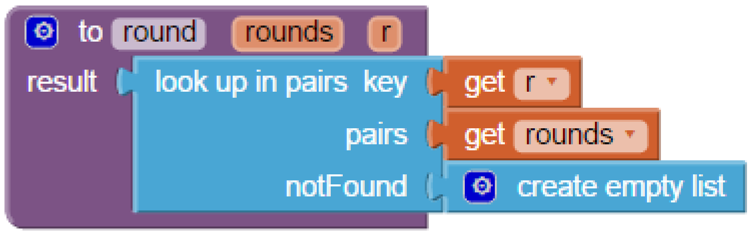
get_winning_moves



```
to get_winning_moves match
result  look up in pairs  key      " winning_moves "
                          pairs    look up in pairs  key      look up in pairs  key      get global game_type_FDB_TAG
                                                                                pairs    get match
                                                                                notFound " "
                                                     pairs    get global Rules
                                                     notFound create empty list
                          notFound " "
```

Called by: playByPlay.

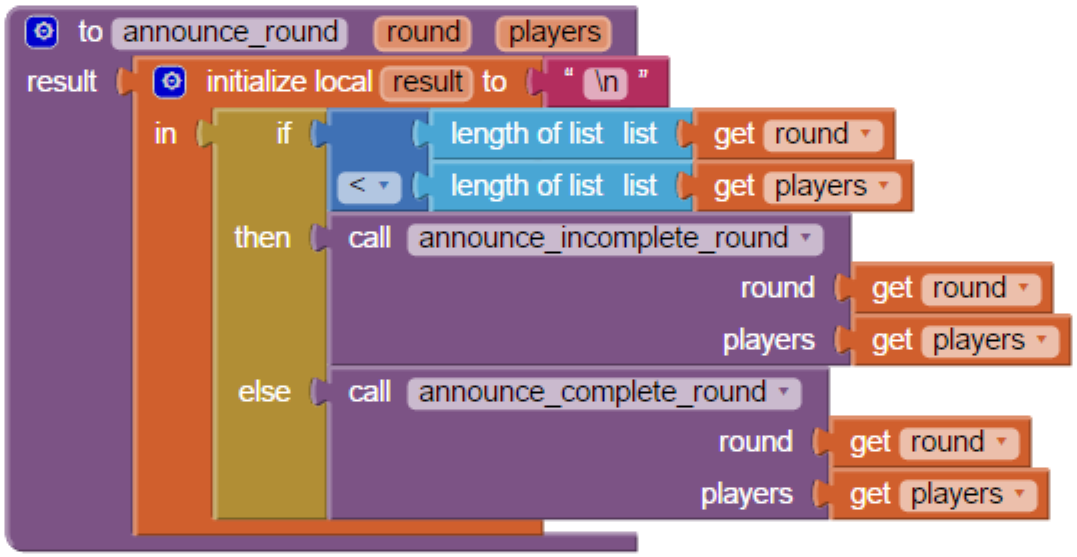
round



```
to round rounds r
result  look up in pairs  key      get r
                          pairs    get rounds
                          notFound create empty list
```

Called by: playByPlay.

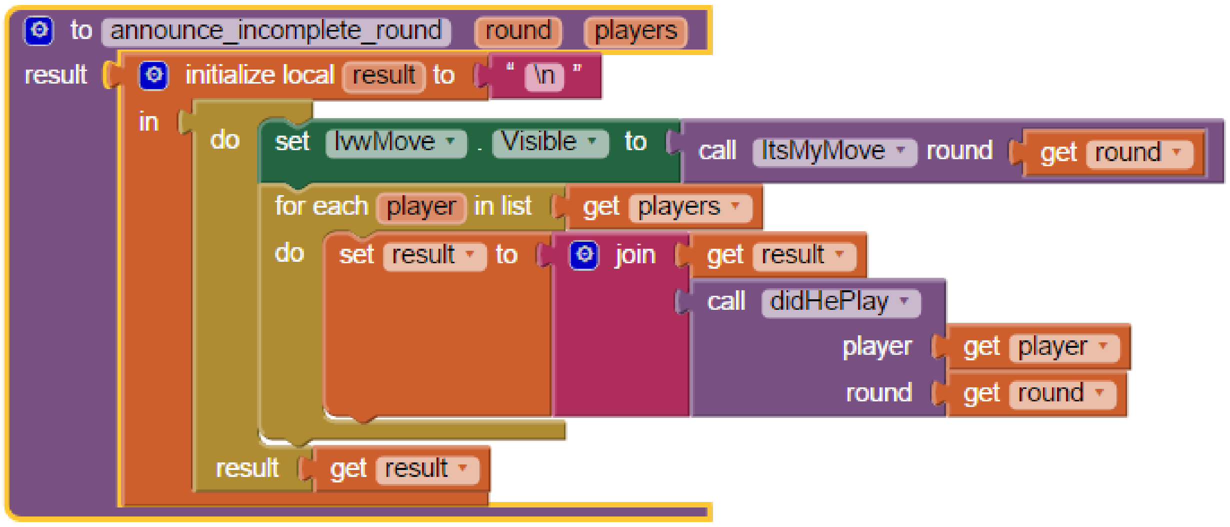
announce_round



```
to announce_round round players
result  initialize local result to " \n "
  in  if    length of list list  get round
          <  length of list list  get players
      then  call announce_incomplete_round
                round    get round
                players  get players
      else  call announce_complete_round
                round    get round
                players  get players
```

Called by: playByPlay.

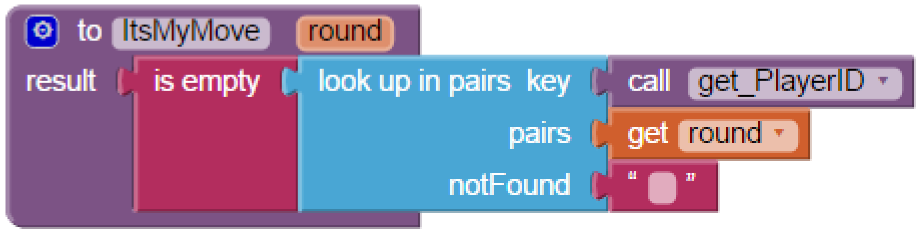
announce_incomplete_round



```
to announce_incomplete_round round players
result  initialize local result to " \n "
  in  do  set IwwMove . Visible to  call ItsMyMove round  get round
          for each player in list  get players
          do  set result to  join  get result
                                   call didHePlay
                                       player  get player
                                       round   get round
      result  get result
```

Called by: playByPlay.

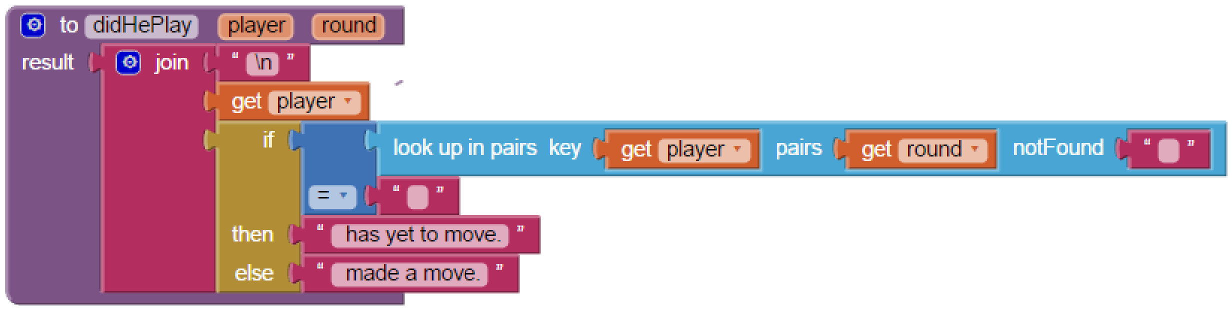
ItsMyMove



```
to ItsMyMove round
result  is empty  look up in pairs  key       call get_PlayerID
                                    pairs     get round
                                    notFound  " "
```

Called by: announce_incomplete_round

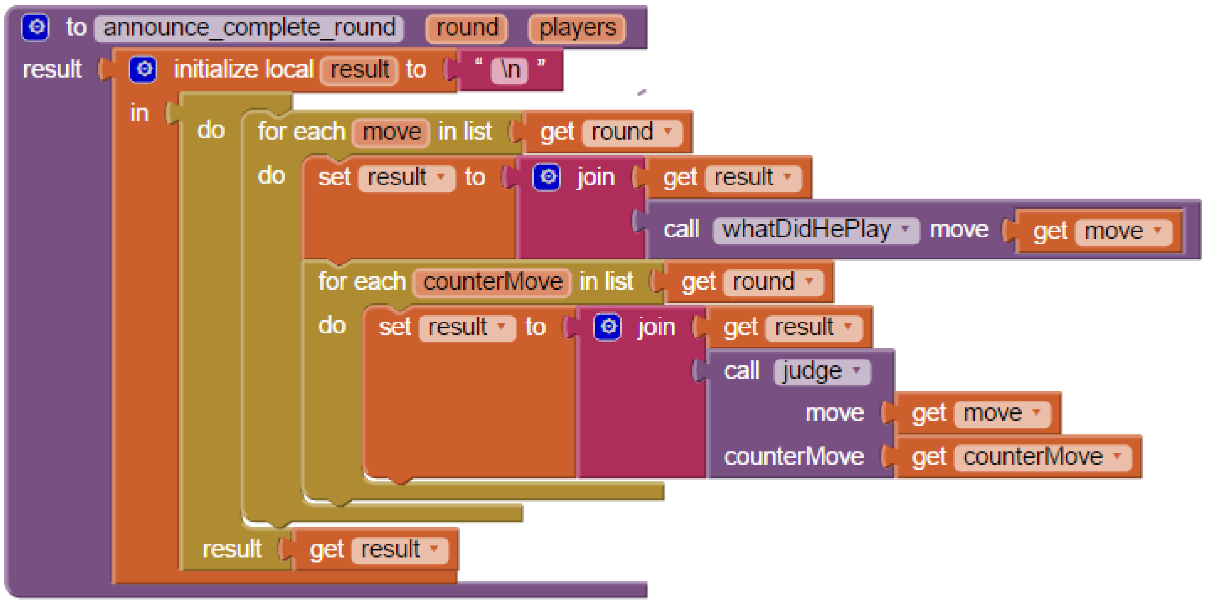
didHePlay



```
to didHePlay player round
result  join  " \n "
              get player
              if    look up in pairs  key get player  pairs get round  notFound " "
                    = " "
              then  " has yet to move. "
              else  " made a move. "
```

Called by: announce_incomplete_round

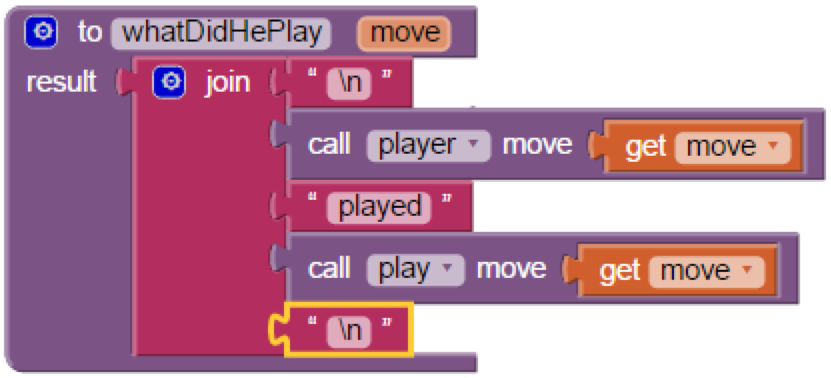
announce_complete_round



```
to announce_complete_round round players
result  initialize local result to " \n "
        in  do  for each move in list get round
                    do  set result to join  get result
                                            call whatDidHePlay move get move
                        for each counterMove in list get round
                            do  set result to join  get result
                                                    call judge
                                                        move get move
                                                        counterMove get counterMove
                result get result
```

Called by: [announce_round]()

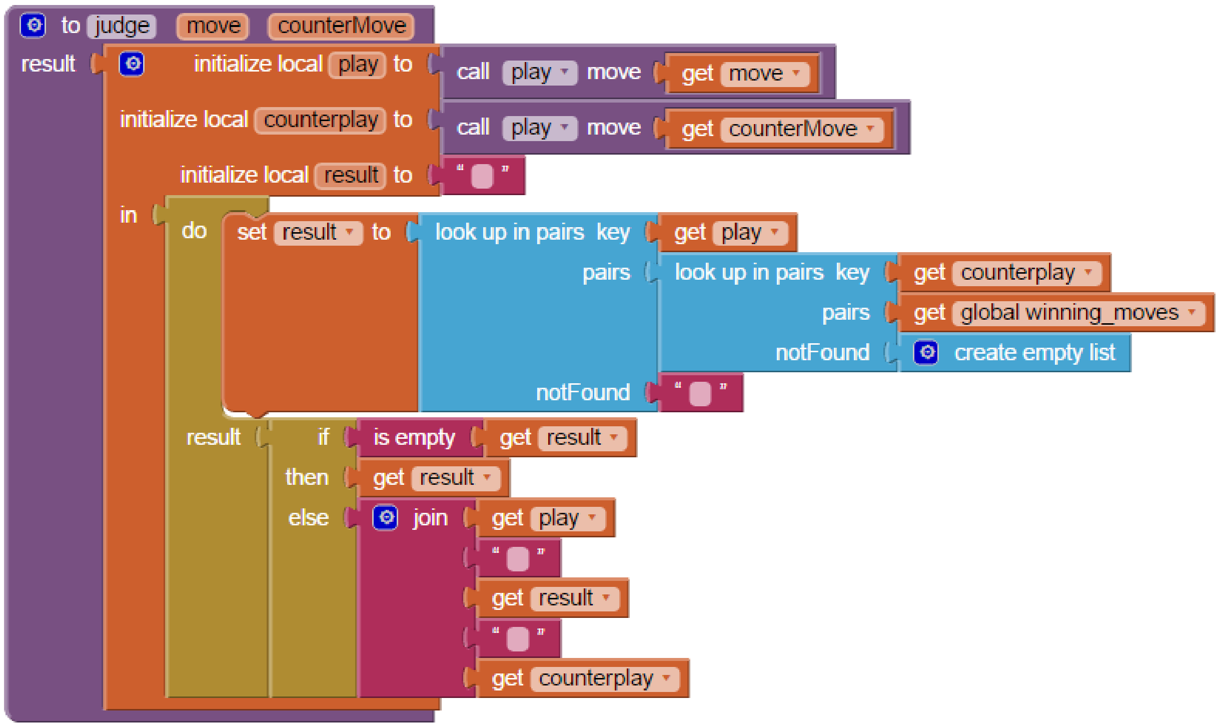
whatDidHePlay



```
to whatDidHePlay move
result  join  " \n "
              call player move get move
              " played "
              call play move get move
              " \n "
```

Called by: [announce_complete_round]()

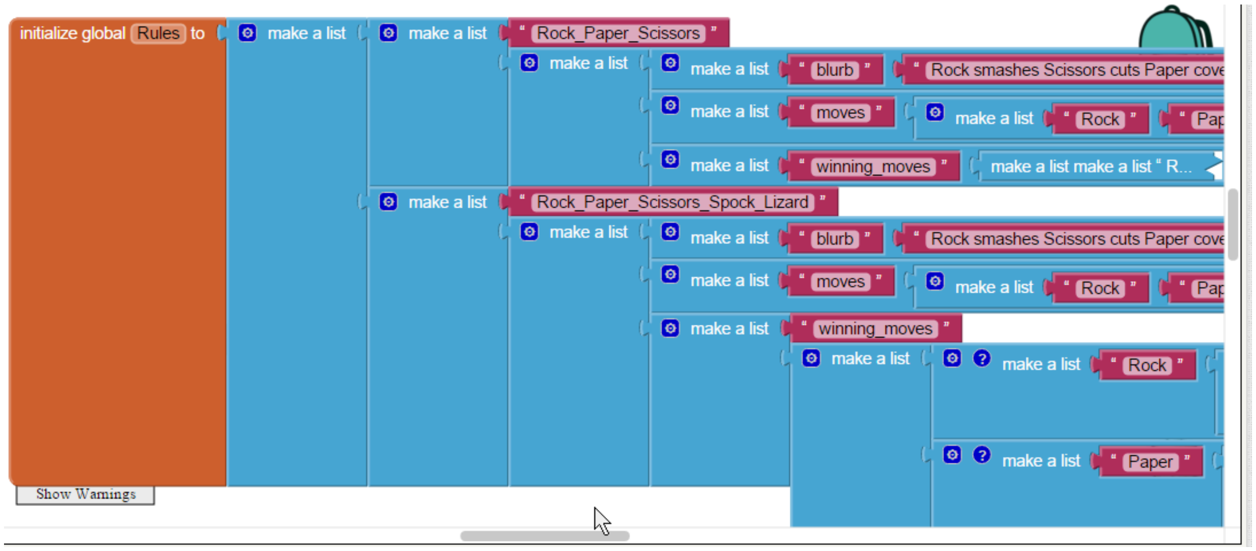
judge



```
to judge move counterMove
result  initialize local play to  call play move  get move
        initialize local counterplay to  call play move  get counterMove
        initialize local result to  " "
        in  do  set result to  look up in pairs  key  get play
                                                 pairs  look up in pairs  key  get counterplay
                                                                          pairs  get global winning_moves
                                                                          notFound  create empty list
                                                 notFound  " "
            result  if    is empty  get result
                    then  get result
                    else  join  get play
                                " "
                                get result
                                " "
                                get counterplay
```

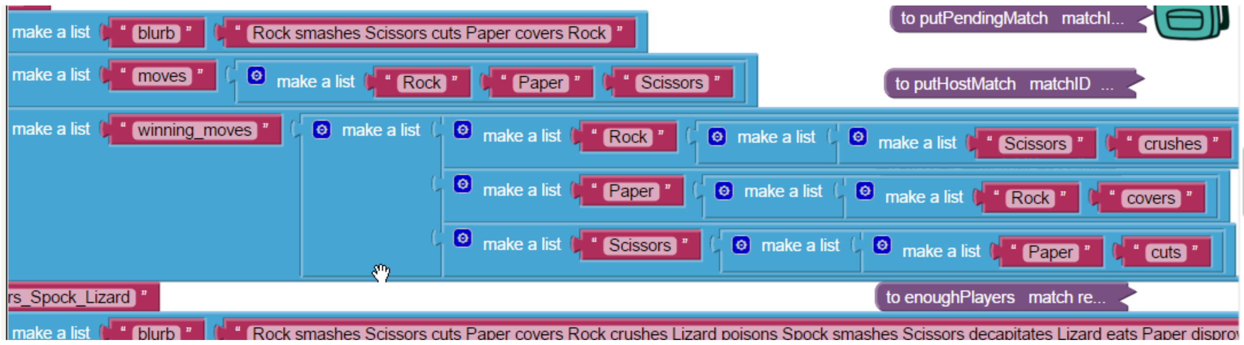
Called by: announce_complete_round

Rules



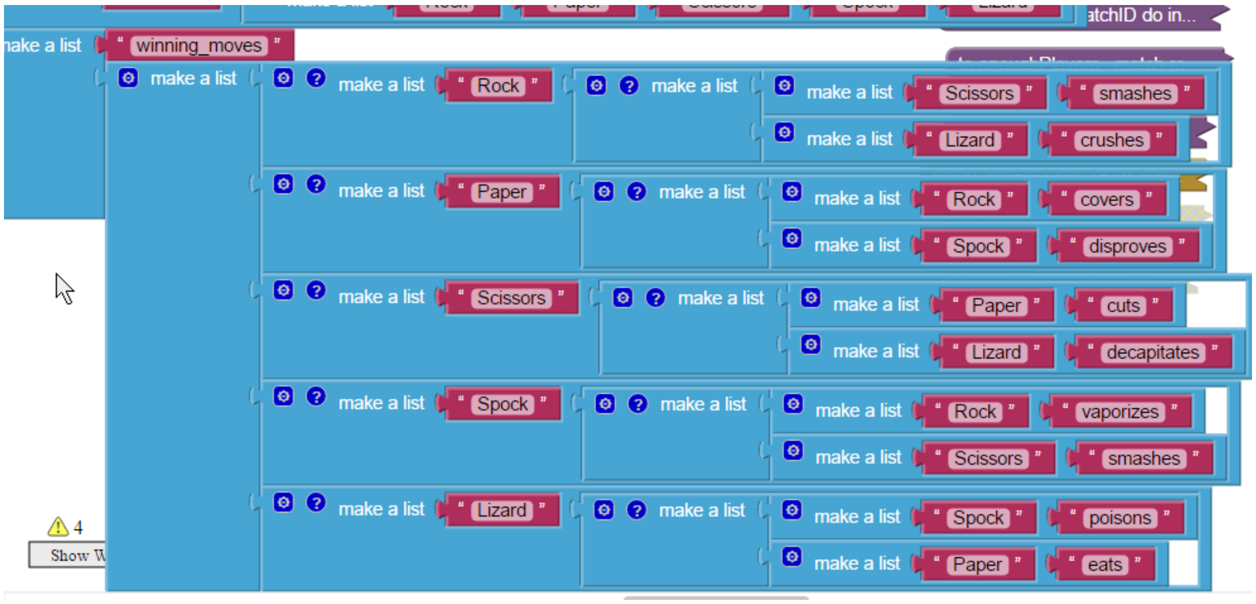
```
initialize global Rules to  make a list  make a list  " Rock_Paper_Scissors "
                                                      make a list  make a list  " blurb "  " Rock smashes Scissors cuts Paper cove
                                                                   make a list  " moves "  make a list  " Rock "  " Pap
                                                                   make a list  " winning_moves "  make a list make a list " R...
                                         make a list  " Rock_Paper_Scissors_Spock_Lizard "
                                                      make a list  make a list  " blurb "  " Rock smashes Scissors cuts Paper cove
                                                                   make a list  " moves "  make a list  " Rock "  " Pap
                                                                   make a list  " winning_moves "
                                                                                make a list  ?  make a list  " Rock "
                                                                                             ?  make a list  " Paper "
Show Warnings
```

Rules - RPS



```
make a list  " blurb "  " Rock smashes Scissors cuts Paper covers Rock "          to putPendingMatch  matchI...
make a list  " moves "  make a list  " Rock "  " Paper "  " Scissors "            to putHostMatch  matchID ...
make a list  " winning_moves "  make a list  make a list  " Rock "  make a list  make a list  " Scissors "  " crushes "
                                             make a list  " Paper "  make a list  make a list  " Rock "  " covers "
                                             make a list  " Scissors "  make a list  make a list  " Paper "  " cuts "
rs_Spock_Lizard "                                                                 to enoughPlayers  match re...
make a list  " blurb "  " Rock smashes Scissors cuts Paper covers Rock crushes Lizard poisons Spock smashes Scissors decapitates Lizard eats Paper dispro
```

Rules - RPSSL

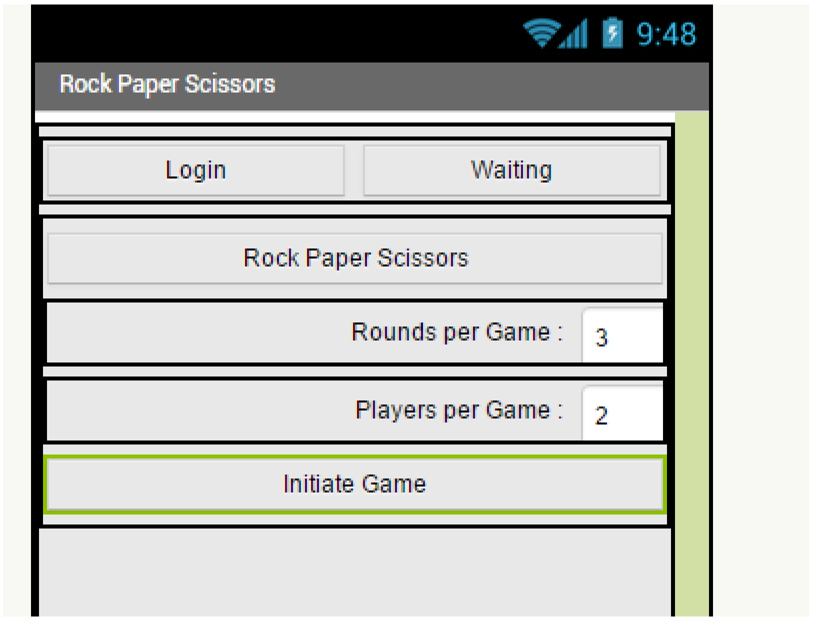


```
make a list "winning_moves"
  make a list
    make a list "Rock"
      make a list
        make a list "Scissors" "smashes"
        make a list "Lizard" "crushes"
    make a list "Paper"
      make a list
        make a list "Rock" "covers"
        make a list "Spock" "disproves"
    make a list "Scissors"
      make a list
        make a list "Paper" "cuts"
        make a list "Lizard" "decapitates"
    make a list "Spock"
      make a list
        make a list "Rock" "vaporizes"
        make a list "Scissors" "smashes"
    make a list "Lizard"
      make a list
        make a list "Spock" "poisons"
        make a list "Paper" "eats"
```

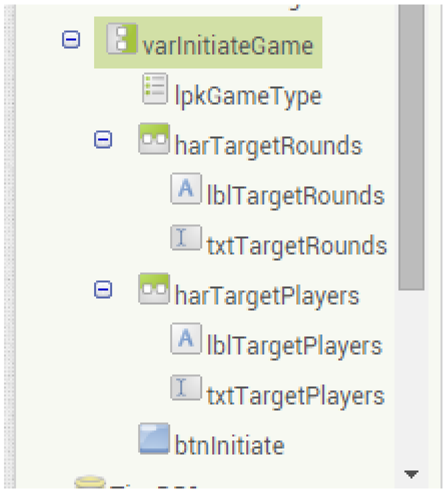
Initiate a match

select game type, target rounds, target wins

Designer layout - Initiate a match

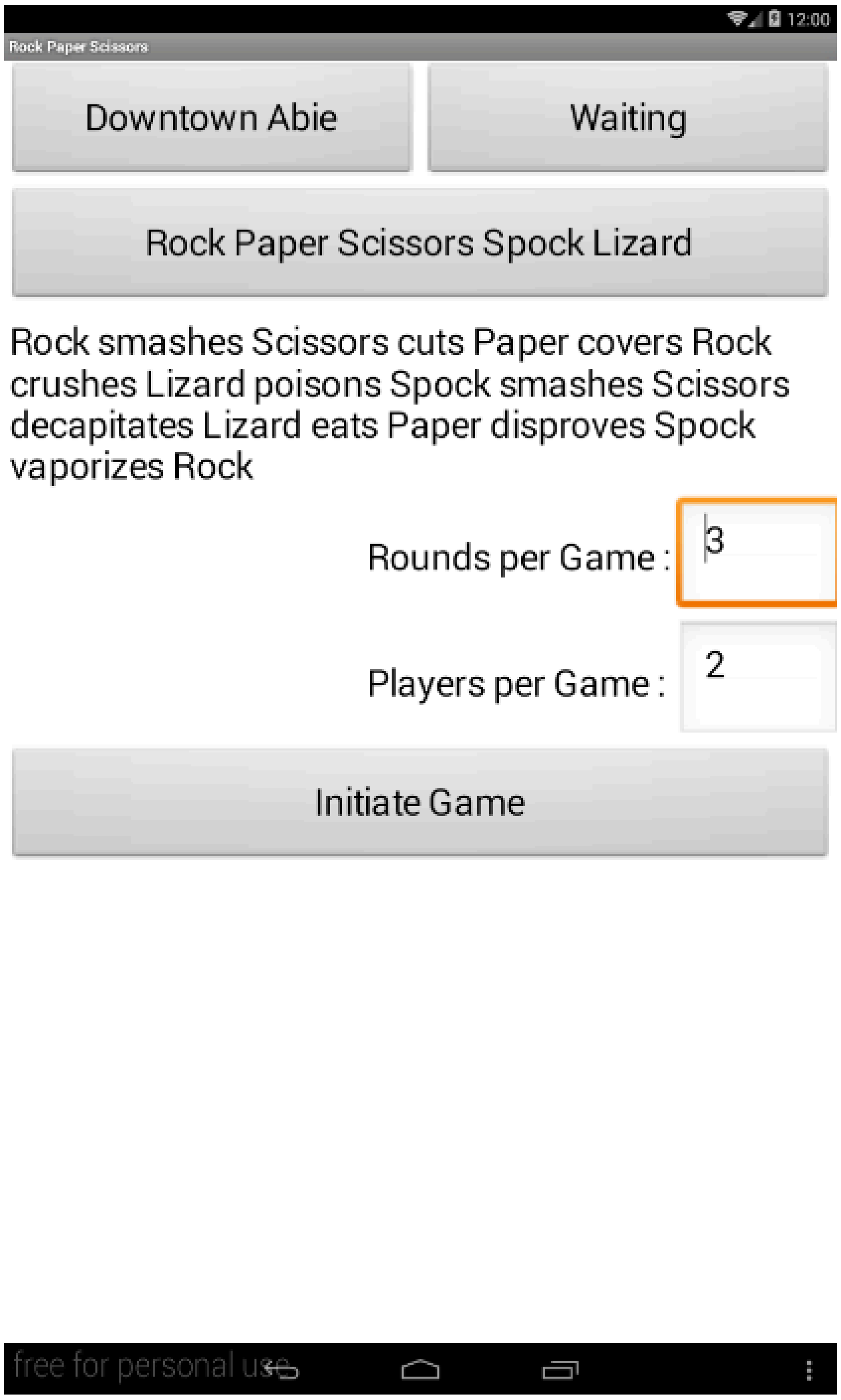


Designer components - Initiate a match



varInitiateGame
lpkGameType
harTargetRounds
lblTargetRounds
txtTargetRounds
harTargetPlayers
lblTargetPlayers
txtTargetPlayers
btnInitiate

Select a Game Type


12:00

Rock Paper Scissors

Downtown Abie

Waiting

Rock Paper Scissors Spock Lizard

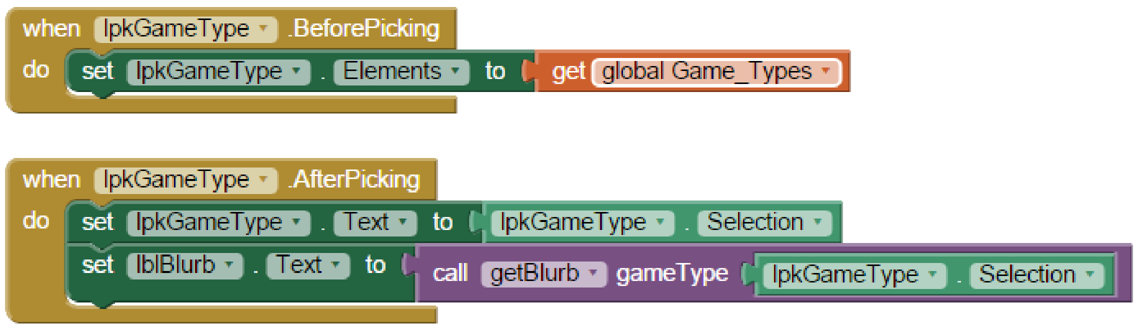
Rock smashes Scissors cuts Paper covers Rock crushes Lizard poisons Spock smashes Scissors decapitates Lizard eats Paper disproves Spock vaporizes Rock

Rounds per Game : 3

Players per Game : 2

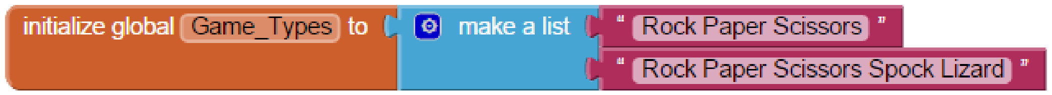
Initiate Game

Game Type List Picker blocks



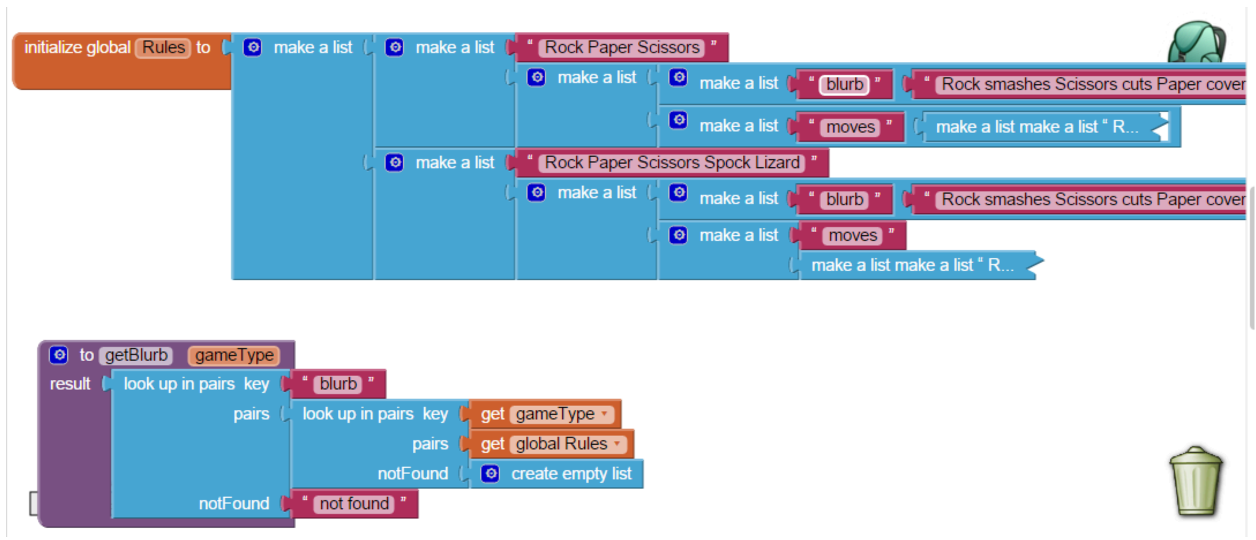
```
when lpkGameType .BeforePicking
do  set lpkGameType . Elements to  get global Game_Types

when lpkGameType .AfterPicking
do  set lpkGameType . Text to  lpkGameType . Selection
    set lblBlurb . Text to  call getBlurb gameType  lpkGameType . Selection
```



```
initialize global Game_Types to  make a list  " Rock Paper Scissors "
                                              " Rock Paper Scissors Spock Lizard "
```

Game Type Blurb Lookup

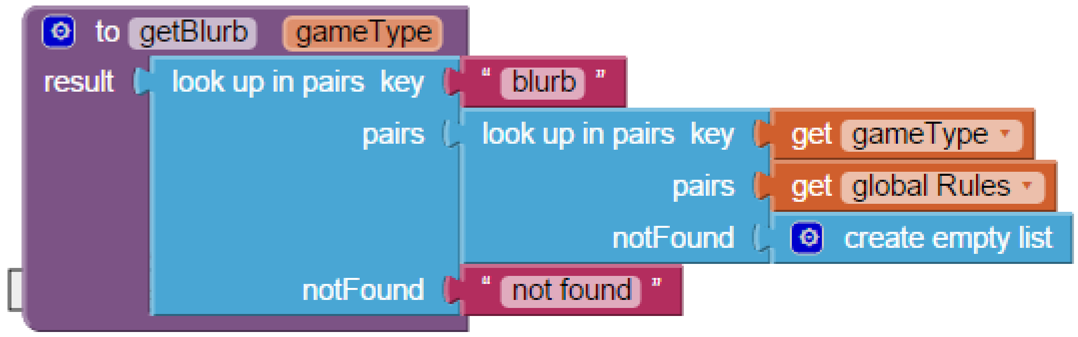


```
initialize global Rules to  make a list  make a list  " Rock Paper Scissors "
                                                      make a list  make a list  " blurb "  " Rock smashes Scissors cuts Paper cover
                                                                   make a list  " moves "  make a list make a list " R...
                                         make a list  " Rock Paper Scissors Spock Lizard "
                                                      make a list  make a list  " blurb "  " Rock smashes Scissors cuts Paper cover
                                                                   make a list  " moves "
                                                                                make a list make a list " R...

to getBlurb gameType
result  look up in pairs  key  " blurb "
                          pairs  look up in pairs  key  get gameType
                                                   pairs  get global Rules
                                                   notFound  create empty list
                          notFound  " not found "
```

Game Type blurbs tell the moves in English. They are stored with the rules for each game type in a list of pairs structure designed for use with the **lookup in pairs** block.

The Blurb lookup is packaged in a value procedure, because it will probably happen again when it's time to choose a move. Notice how the lookup proceeds in opposite order (inner to outer) to the nesting of the lookup table (Rules).


```
to getBlurb gameType
result  look up in pairs key  " blurb "
                         pairs  look up in pairs key  get gameType
                                                 pairs  get global Rules
                                              notFound  create empty list
                      notFound  " not found "
```

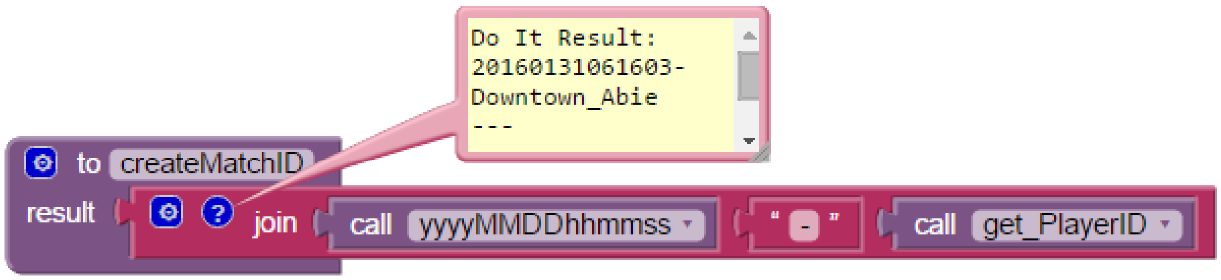
Generating a unique gameID for a new game

(See createMatchID)

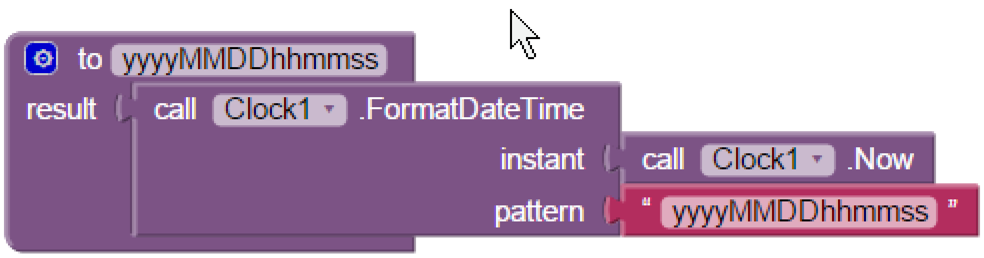
For lack of a proper guid provider, we use a sortable datetimestamp combined with the current PlayerID, which should be unique.

We will use this as the new current match ID, accessed through set and get procedures:

createMatchID

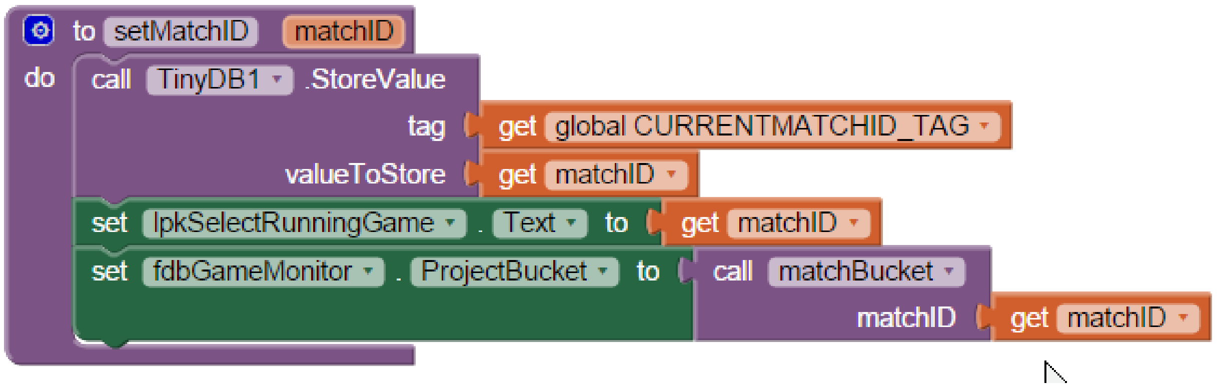

```
Do It Result:
20160131061603-
Downtown_Abie
---
```
```
to createMatchID
result  join  call yyyyMMDDhhmmss   " - "   call get_PlayerID
```

yyyyMMDDhhmmss



```
to yyyyMMDDhhmmss
result  call Clock1 .FormatDateTime
                         instant  call Clock1 .Now
                         pattern  " yyyyMMDDhhmmss "
```

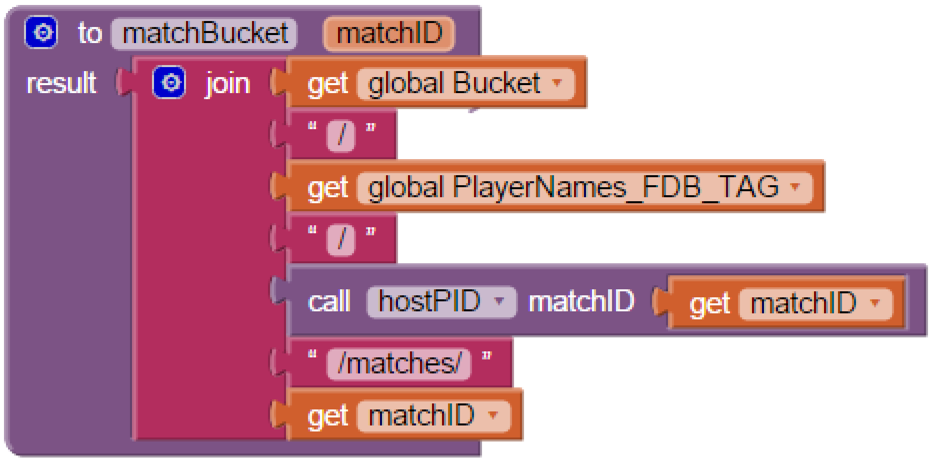
setMatchID



```
to setMatchID  matchID
do  call TinyDB1 .StoreValue
                       tag  get global CURRENTMATCHID_TAG
              valueToStore  get matchID
    set lpkSelectRunningGame . Text to  get matchID
    set fdbGameMonitor . ProjectBucket to  call matchBucket
                                                   matchID  get matchID
```

To set the current match, we update TinyDB, identify it on the Text of it ListPicker, and set the project bucket of the FirebaseDB Game Monitor to watch the match in its owner’s matches area.

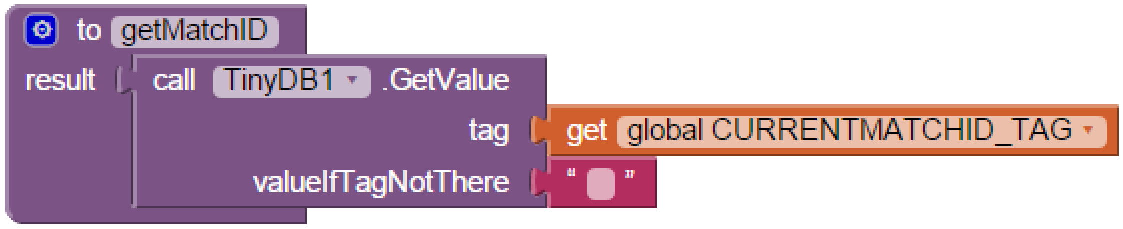
matchBucket



```
to matchBucket matchID
result  join  get global Bucket
              "/"
              get global PlayerNames_FDB_TAG
              "/"
              call hostPID  matchID  get matchID
              "/matches/"
              get matchID
```

This bucket value homes in our db monitor to the current match.

getMatchID

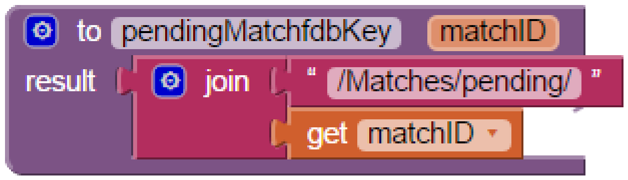


```
to getMatchID
result  call TinyDB1 .GetValue
                         tag  get global CURRENTMATCHID_TAG
            valueIfTagNotThere  " "
```

Saving a pending match to FireBaseDB

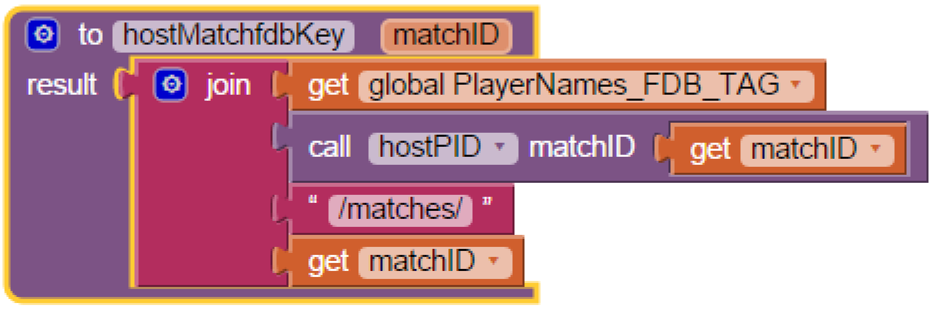
pendingMatchfdbKey

The pending matches FireBaseDB path, based on RPSSL.



```
to pendingMatchfdbKey matchID
result  join  "/Matches/pending/"
              get matchID
```

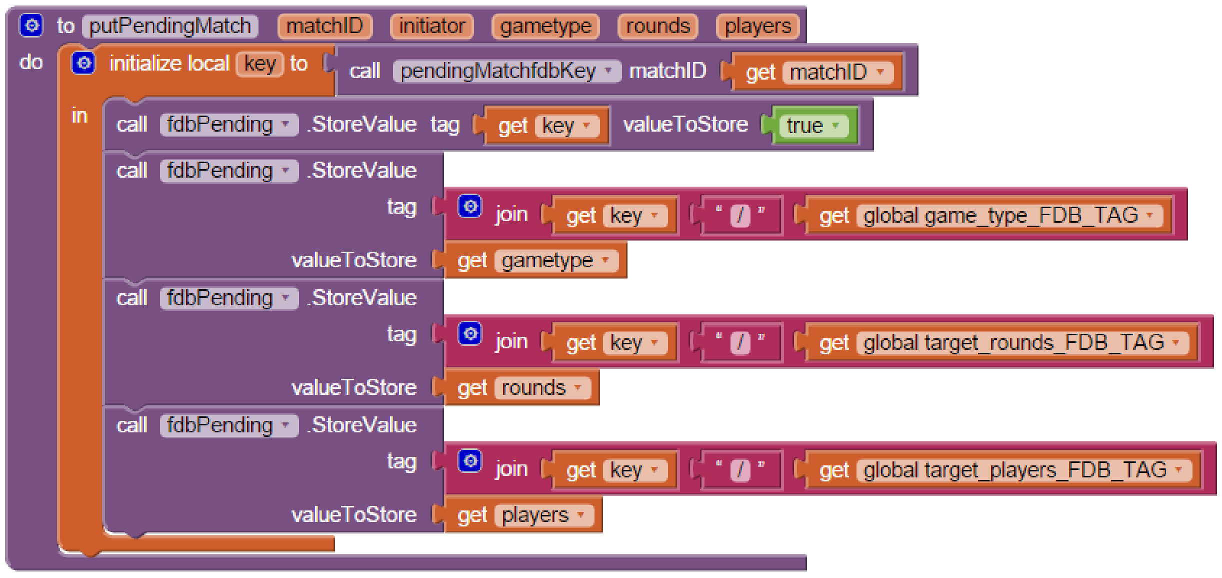
hostMatchFDBKey



```
to hostMatchfdbKey matchID
result  join  get global PlayerNames_FDB_TAG
              call hostPID matchID  get matchID
              " /matches/ "
              get matchID
```

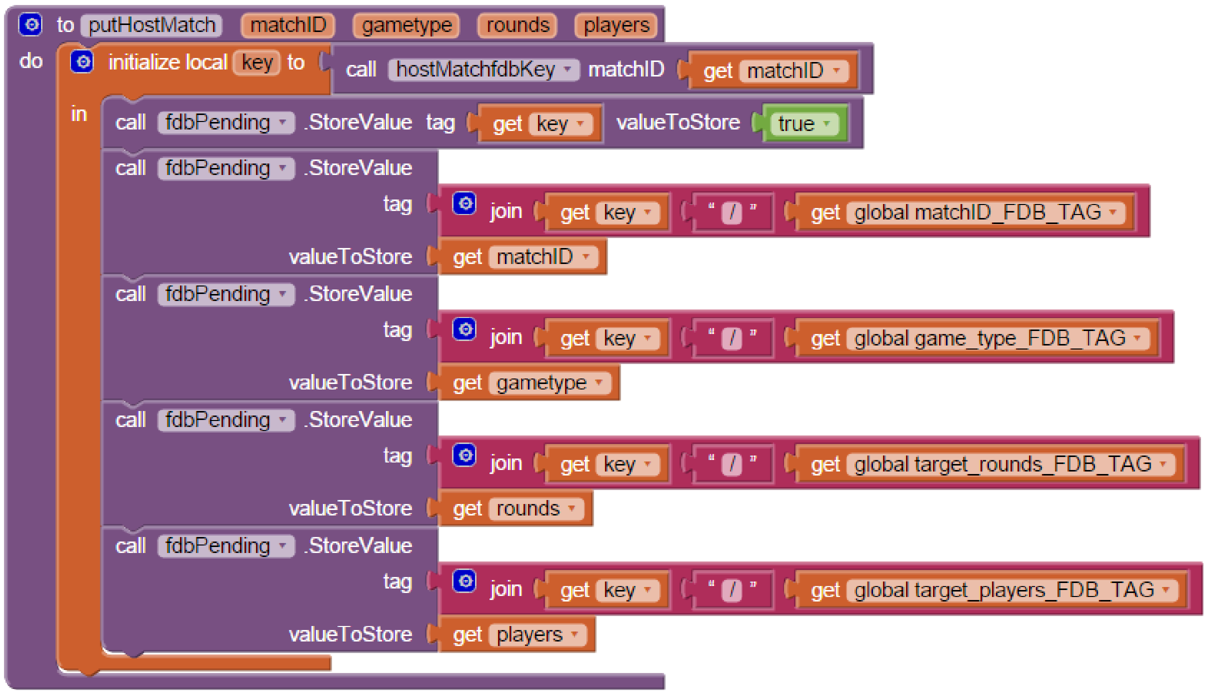
This assumes a base bucket of RPSSL.

putPendingMatch



```
to putPendingMatch matchID initiator gametype rounds players
do  initialize local key to  call pendingMatchfdbKey matchID  get matchID
    in  call fdbPending .StoreValue tag  get key  valueToStore  true
        call fdbPending .StoreValue
                   tag  join  get key  " / "  get global game_type_FDB_TAG
          valueToStore  get gametype
        call fdbPending .StoreValue
                   tag  join  get key  " / "  get global target_rounds_FDB_TAG
          valueToStore  get rounds
        call fdbPending .StoreValue
                   tag  join  get key  " / "  get global target_players_FDB_TAG
          valueToStore  get players
```

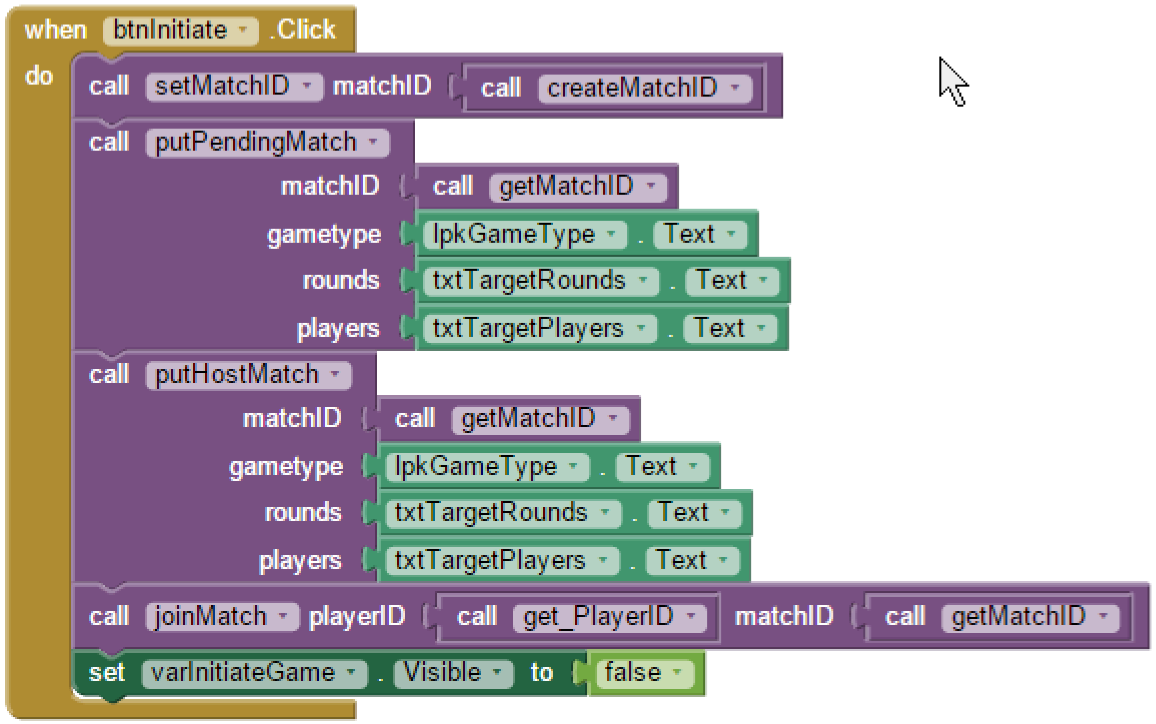
For each pending match, we store its game type, target rounds, and target players, all under its matchID in the Matches/pending/ branch. references: pendingMatchfdbKey putHostMatch



```
to putHostMatch matchID gametype rounds players
do  initialize local key to  call hostMatchfdbKey matchID  get matchID
    in  call fdbPending .StoreValue tag  get key  valueToStore  true
        call fdbPending .StoreValue
                                tag  join  get key
                                           " / "
                                           get global matchID_FDB_TAG
                       valueToStore  get matchID
        call fdbPending .StoreValue
                                tag  join  get key
                                           " / "
                                           get global game_type_FDB_TAG
                       valueToStore  get gametype
        call fdbPending .StoreValue
                                tag  join  get key
                                           " / "
                                           get global target_rounds_FDB_TAG
                       valueToStore  get rounds
        call fdbPending .StoreValue
                                tag  join  get key
                                           " / "
                                           get global target_players_FDB_TAG
                       valueToStore  get players
```

references: hostMatchFDBKey

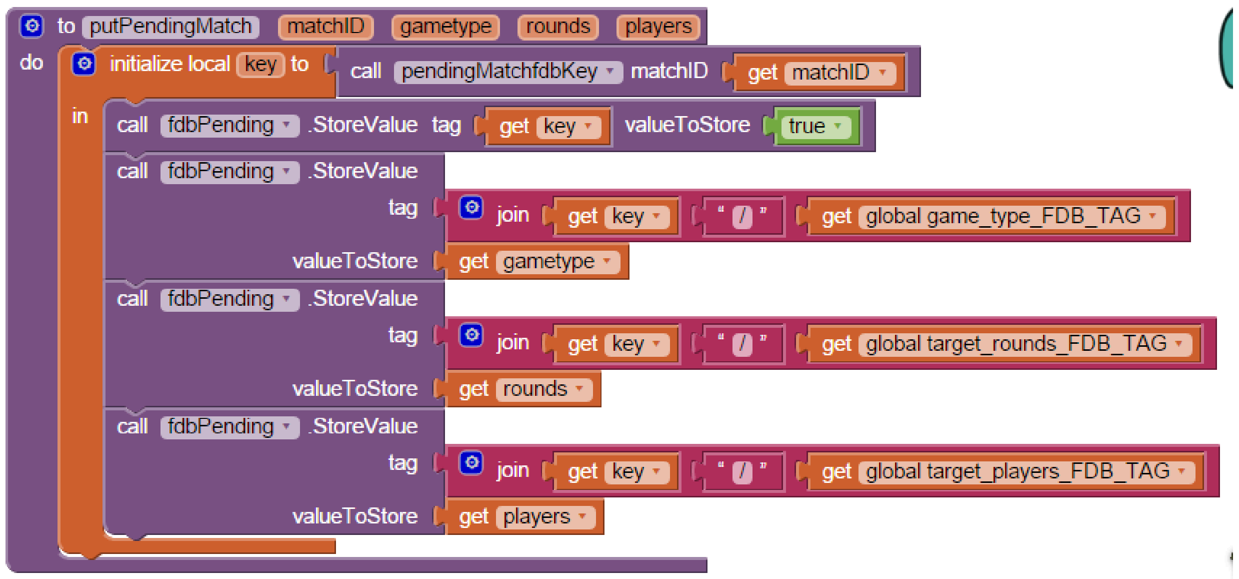
btnInitiate



```
when btnInitiate .Click
do  call setMatchID matchID  call createMatchID
    call putPendingMatch
                     matchID  call getMatchID
                     gametype lpkGameType . Text
                     rounds   txtTargetRounds . Text
                     players  txtTargetPlayers . Text
    call putHostMatch
                     matchID  call getMatchID
                     gametype lpkGameType . Text
                     rounds   txtTargetRounds . Text
                     players  txtTargetPlayers . Text
    call joinMatch playerID  call get_PlayerID  matchID  call getMatchID
    set varInitiateGame . Visible to false
```

A pending match is stored temporarily in the Matches/pending section as well as under the matches subsection of the initiating player. References: getMatchID, createMatchID, putPendingMatch, putHostMatch, get_PlayerID, joinMatch

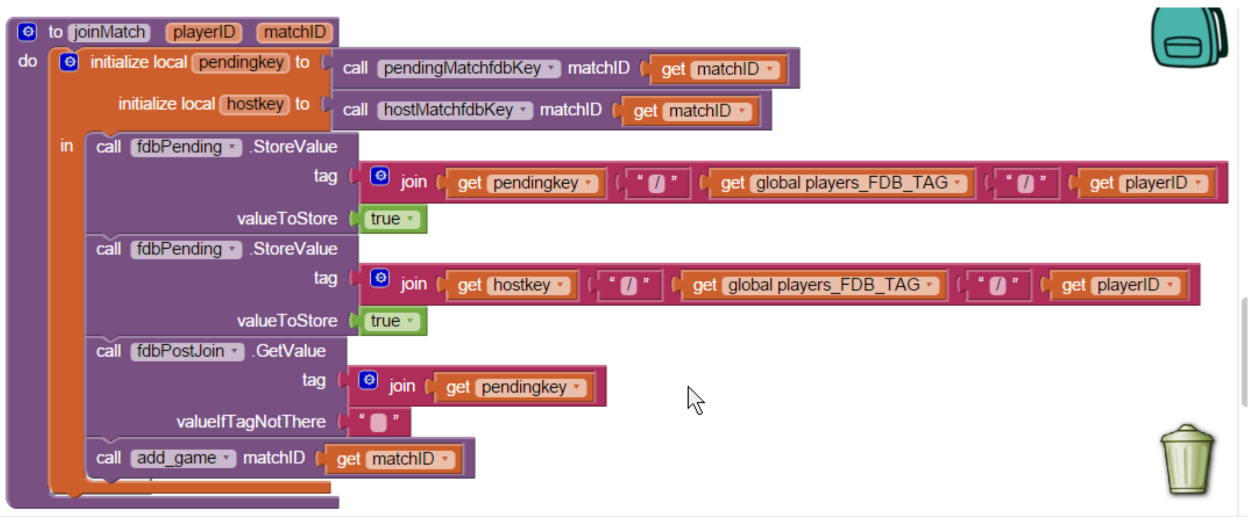
putPendingMatch



```
to putPendingMatch matchID gametype rounds players
do  initialize local key to  call pendingMatchfdbKey matchID  get matchID
    in  call fdbPending.StoreValue tag  get key  valueToStore  true
        call fdbPending.StoreValue
                          tag  join  get key
                                     " / "
                                     get global game_type_FDB_TAG
                 valueToStore  get gametype
        call fdbPending.StoreValue
                          tag  join  get key
                                     " / "
                                     get global target_rounds_FDB_TAG
                 valueToStore  get rounds
        call fdbPending.StoreValue
                          tag  join  get key
                                     " / "
                                     get global target_players_FDB_TAG
                 valueToStore  get players
```

References: pendingMatchfdbKey

joinMatch

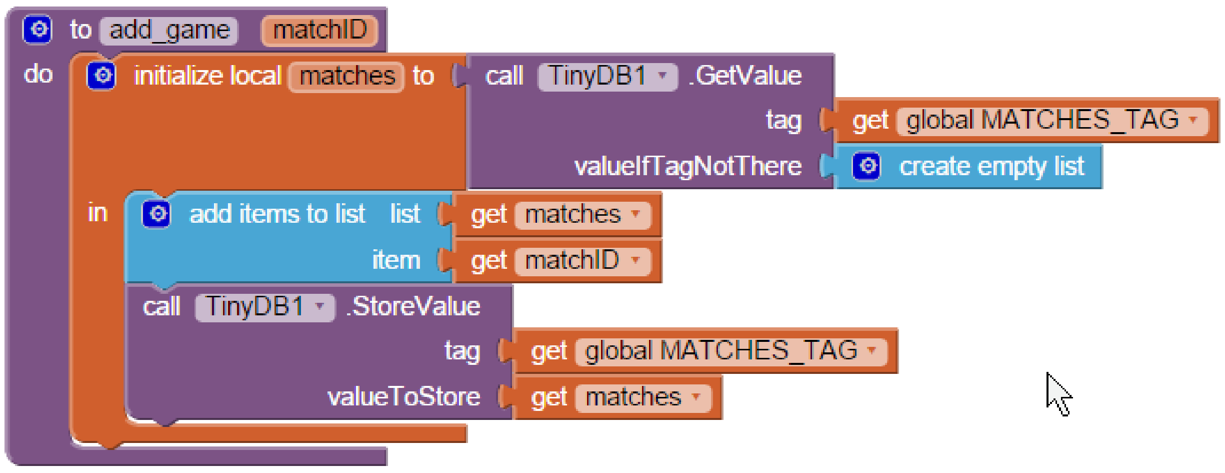


```
to joinMatch playerID matchID
do  initialize local pendingkey to  call pendingMatchfdbKey matchID  get matchID
        initialize local hostkey to  call hostMatchfdbKey matchID  get matchID
    in  call fdbPending.StoreValue
                          tag  join  get pendingkey
                                     " / "
                                     get global players_FDB_TAG
                                     " / "
                                     get playerID
                 valueToStore  true
        call fdbPending.StoreValue
                          tag  join  get hostkey
                                     " / "
                                     get global players_FDB_TAG
                                     " / "
                                     get playerID
                 valueToStore  true
        call fdbPostJoin.GetValue
                          tag  join  get pendingkey
              valueIfTagNotThere  " "
        call add_game matchID  get matchID
```

Joining a match is done both at the pending match section and at the home copy of the match under the originating player, and in TinyDB1 tag MATCHES. After joining the match, we double check if the match has enough players to commence in fdbPostJoin.GotValue.
References: pendingMatchfdbKey, hostMatchfdbKey, add_game

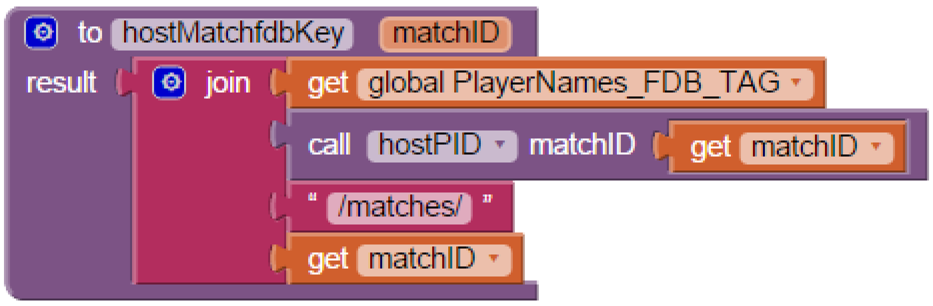
add_game

This adds a matchID to the app’s TinyDB list of matches, for display in the matchID list picker.



```
to add_game matchID
do  initialize local matches to  call TinyDB1 .GetValue
                                           tag  get global MATCHES_TAG
                             valueIfTagNotThere  create empty list
    in  add items to list  list  get matches
                           item  get matchID
        call TinyDB1 .StoreValue
                       tag  get global MATCHES_TAG
               valueToStore  get matches
```

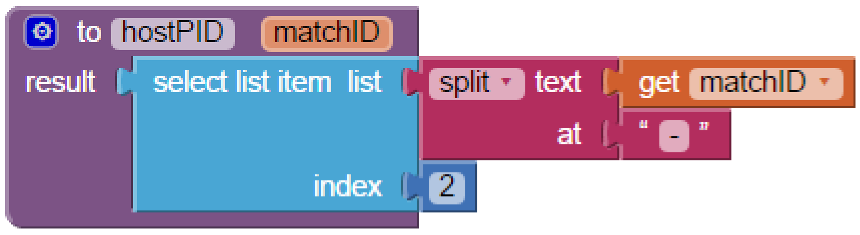
hostMatchfdbKey



```
to hostMatchfdbKey matchID
result  join  get global PlayerNames_FDB_TAG
              call hostPID matchID  get matchID
              " /matches/ "
              get matchID
```

Reference:
hostPID

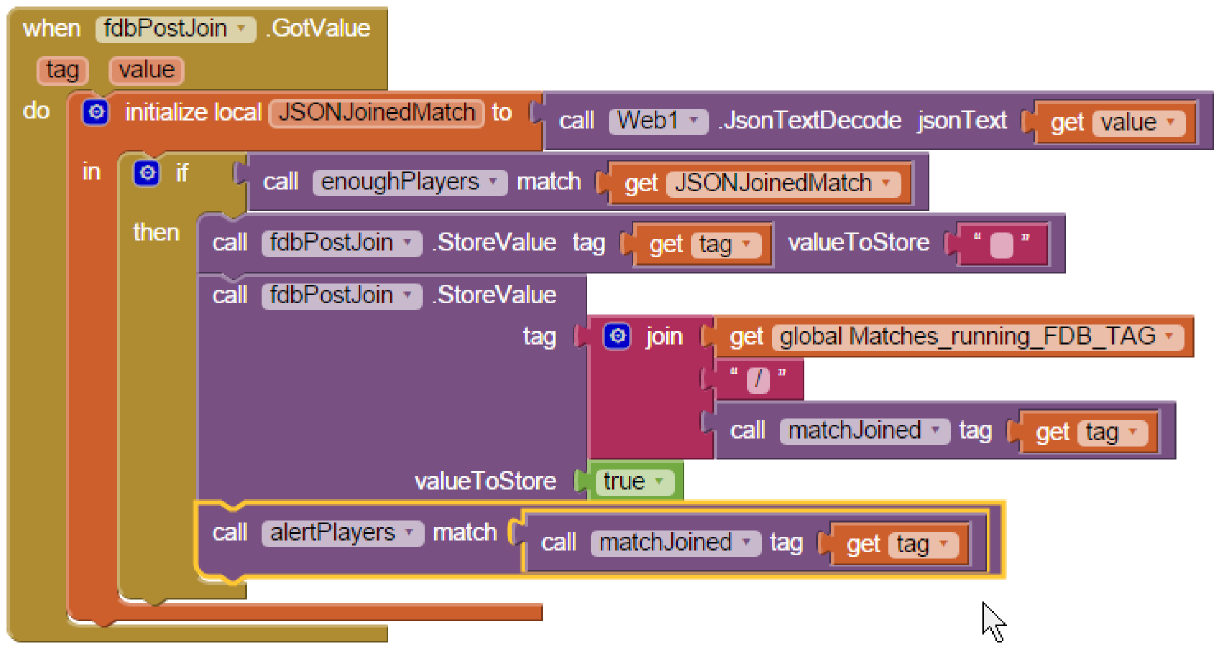
hostPID



```
to hostPID matchID
result  select list item  list  split  text  get matchID
                                          at  " - "
                         index  2
```

The host player ID of a match is kept after the “-” in the matchID.

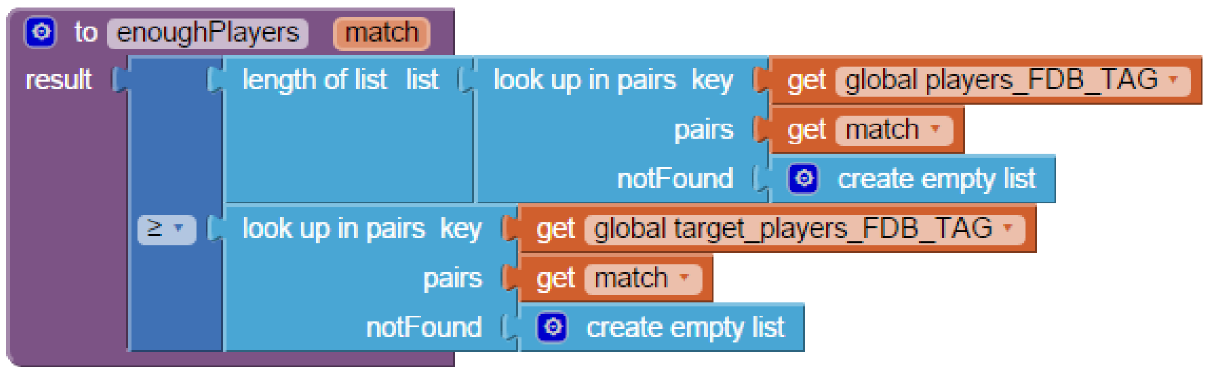
fdbPostJoin.GotValue



```
when fdbPostJoin .GotValue
  tag  value
do  initialize local JSONJoinedMatch to  call Web1 .JsonTextDecode jsonText  get value
    in  if  call enoughPlayers match  get JSONJoinedMatch
        then  call fdbPostJoin .StoreValue tag  get tag  valueToStore  " "
              call fdbPostJoin .StoreValue
                                tag  join  get global Matches_running_FDB_TAG
                                           " / "
                                           call matchJoined tag  get tag
                       valueToStore  true
              call alertPlayers match  call matchJoined tag  get tag
```

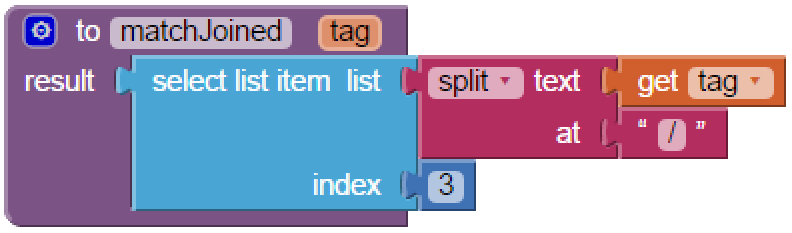
After joining a match, the match is checked if it has enough players to start its first round. The newly looked up match is gotten from Firebase as the entire JSON string under the joined pending matchID. The JSON is decoded into a tree rooted at the matchID. If enough players have joined the match, the pending match is erased from the Matches/pending leg, and added to the Matches/running leg for people who want to watch matches in progress. All the players in this match are alerted by updating their newsfeed timestamp for this match. References: enoughPlayers, matchJoined , alertPlayers,

enoughPlayers



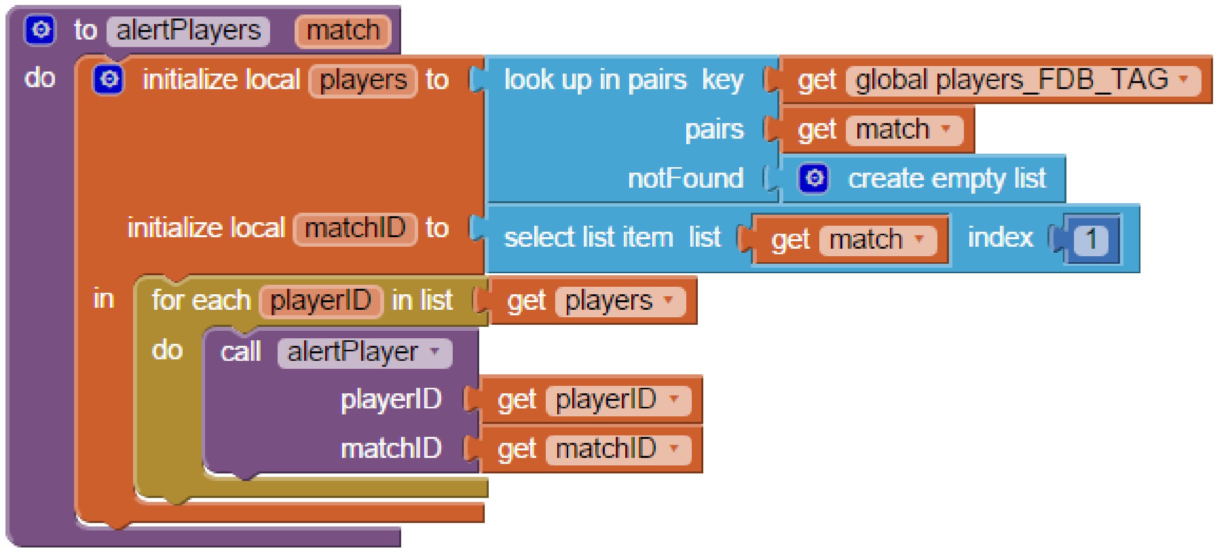
```
to enoughPlayers match
result   length of list list  look up in pairs key  get global players_FDB_TAG
                                               pairs  get match
                                            notFound  create empty list
       ≥ 
         look up in pairs key  get global target_players_FDB_TAG
                        pairs  get match
                     notFound  create empty list
```

matchJoined



```
to matchJoined tag
result  select list item  list  split text  get tag
                                      at  "/"
                          index  3
```

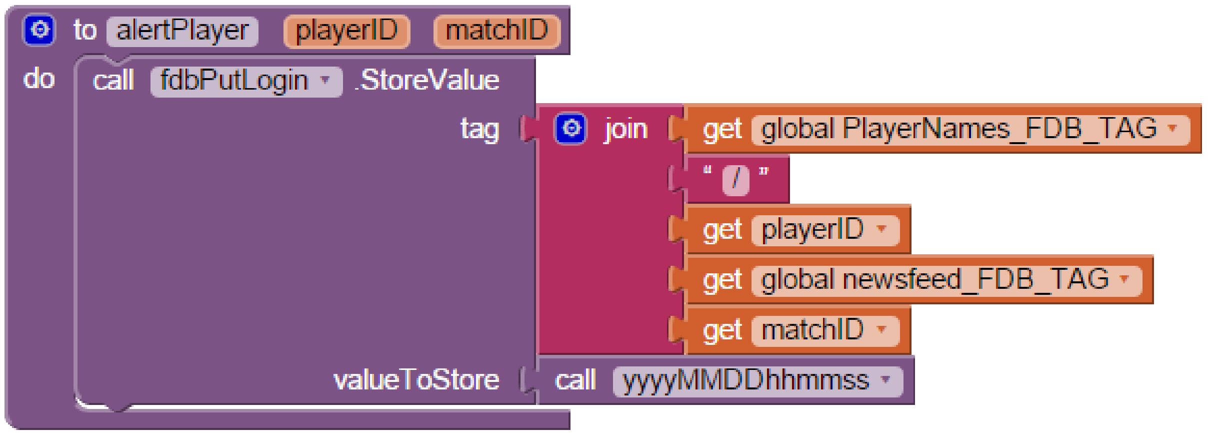
alertPlayers



```
to alertPlayers match
do  initialize local players to  look up in pairs  key  get global players_FDB_TAG
                                                   pairs  get match
                                                   notFound  create empty list
    initialize local matchID to  select list item  list  get match  index  1
    in  for each playerID in list  get players
        do  call alertPlayer
                playerID  get playerID
                matchID  get matchID
```

alertPlayer

This serves to alert a player by inserting a matchID and timestamp into his newsfeed.



```
to alertPlayer playerID matchID
do  call fdbPutLogin .StoreValue
                         tag  join  get global PlayerNames_FDB_TAG
                                    " / "
                                    get playerID
                                    get global newsfeed_FDB_TAG
                                    get matchID
                valueToStore  call yyyyMMDDhhmmss
```

make first move

Review closed matches

- delete closed matches

quit