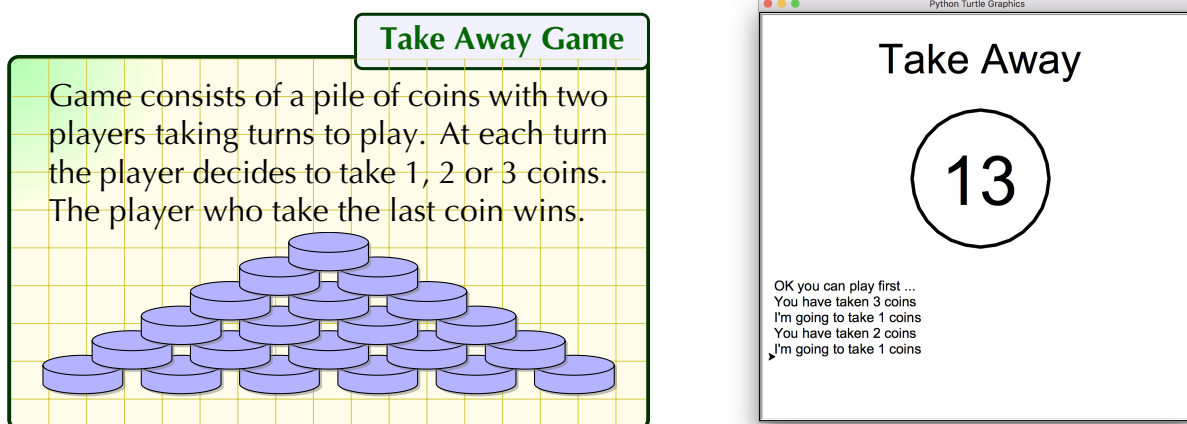


# Graphical Take Away (Second Time Lucky)

## Introduction

In worksheet 2 we developed the *Take Away Game* and we started in worksheet 3 with a graphical version of this game. We covered the various building blocks needed for this but we never got to build the actual game. Let's try to get that completed today.



**Figure 1** – Description of game and output from graphical implementation.

This will be one of the longest programs we will have written so far, and we will need to use the ideas from earlier worksheets so be prepared to look over the earlier worksheets! As before we have broken up the task in separate stages:

### ① Events — We do Nothing unless Prompted!

Remember back in worksheet 3, we talked about *Event Programming*. In event programming we give the computer a set of instructions that look like

“If THIS happens, then do THAT”

Where, for us the THIS will be various keys being pressed.

### ② States — Writing Programs that are Moody

When an event occurs our program may need to do different things depending on what is happening with rest of the programs. We manage this using states.

### ③ Tying it all Together

Next we will implement the game.

### ④ Implementing the GUI

Finally!! we get to the long awaited and overdue graphics !!

# 1 Events — We do Nothing unless Prompted!



Create a new file called `Events.py` and insert the following code.

Run code and see what happens ...

Events.py

```
1 import turtle
2
3 screen = turtle.Screen()
4 bob = turtle.Turtle()
5
6 def startGame():
7     print ("startGame not done yet!")
8
9 screen.onkey(startGame, "s")
10
11 screen.listen()
```

It is boring right? Nothing happens, the computer just sits there waiting and waiting and waiting ...

So what is the code doing?

- In lines 1 to 3, we import the `turtle` library and create a screen for our turtle to run around in and create a turtle, which we call `bob`.
- In lines 6 and 7, we define a simple function that, for now, just outputs a message.
- In line 9 is where the magic happens. Here we tell the screen that if it should “hear” the key “s” being pressed then it should stop whatever it was doing and start running the code in function `startGame`.
- Finally in line 11, we tell the screen to start listening — so it has no excuse if it does not respond to me pressing the “s” key now!



Create a new file called `Events.py` and insert the following code.

- Run the above code and press “s” a few times. What happens?
  - What happens if you press other keys?
  - What happens if you press “S”?
- Fix the program so that it will also output message when user presses “S”.

Ok now that we are happy with how events are programmed, we need to think about what events we will need for our game:

- During the game we need to listen out for keys “1”, “2”, and “3” being pressed — they will represent the number of coins the player wants to remove.
- Also it would be nice if we could start a new at any point — we will use “s” for that (as in our first program above).
- Finally to implement an idea that I stole from Amy, it would be nice at the start of the game to ask the player if they wanted to go first or second. So we need to listen for events “n” and “y” also.



Save your `Events.py` as `TakeAway_Graphical_Events.py` and modify it so that it responds to all of the required events as follows.

#### TakeAway\_Graphical\_Events.py

```
1 import turtle
2
3 screen = turtle.Screen()
4 bob = turtle.Turtle()
5
6 def startGame():
7     print ("startGame not done yet!")
8
9 def player_1():
10    print ("player_1 not done yet!")
11 def player_2():
12    print ("player_2 not done yet!")
13 def player_3():
14    print ("player_3 not done yet!")
15
16 def player_No():
17    print ("player_No not done yet!")
18 def player_Yes():
19    print ("player_Yes not done yet!")
20
21 screen.onkey(startGame, "s")
22 screen.onkey(player_1, "1")
23 screen.onkey(player_2, "2")
24 screen.onkey(player_3, "3")
25 screen.onkey(player_No, "n")
26 screen.onkey(player_Yes, "y")
27
28 screen.listen()
```



Verify that the above code does respond to each of the required events and can handle both upper and lower case input.

OK, so where are we now ... we have got a program that responds to events — I admit the response is a bit sad, but it is a start — however we do not want our programs to always give the same response to an event. We need our program to be moody! This is the focus of the next section.

Test your code!

As our program gets more complicated, bugs will be harder to find. It is vital you test your code at each step of development.



## 2 States — Writing Programs that are Moody

If we run through in our heads what playing the game would be like we would end up with something like the following structure

**start** Game board is setup and game is about to start.

- Program has asked the player if they want to go first. Currently waiting for an “Y” or “N” event, all other events are ignored.

**human** Playing the game and waiting for human move.

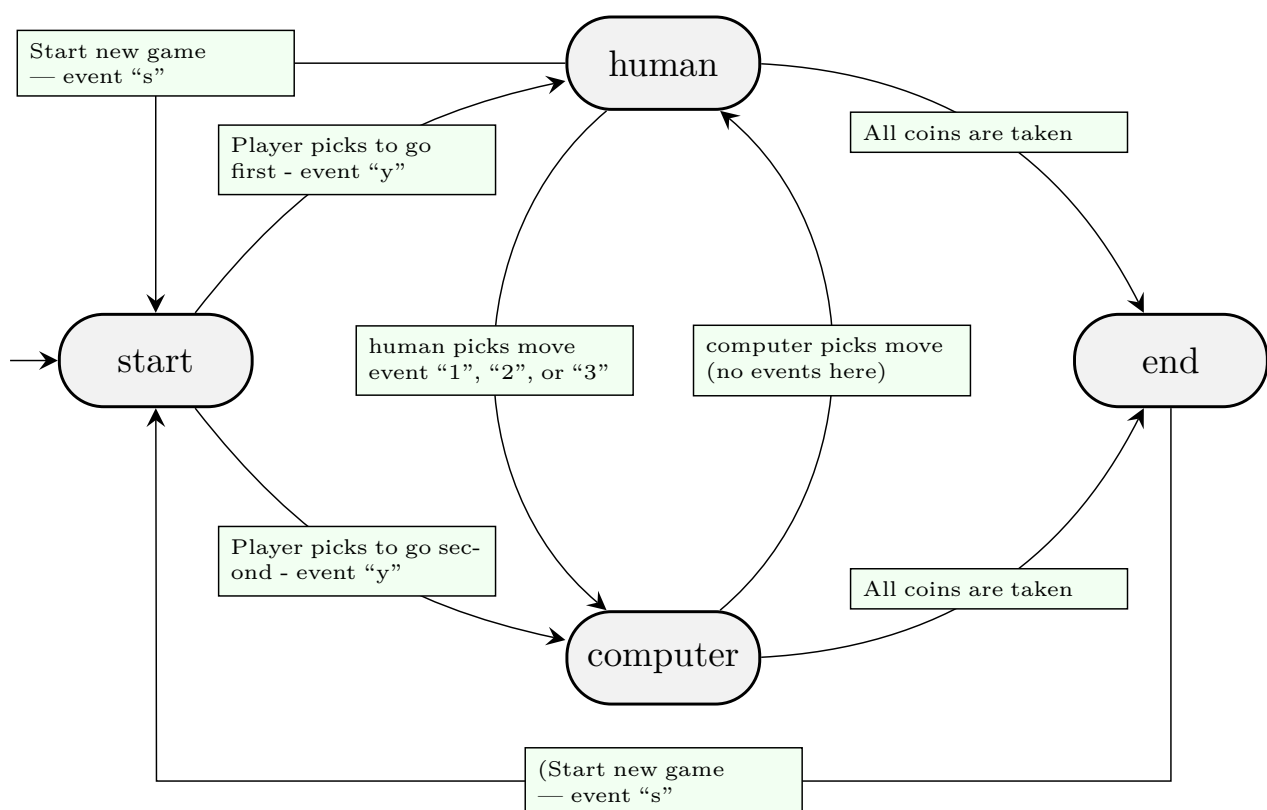
- waiting for human move so events “1”, “2” or “3”.
- Also allowed is “s” to start a new game.

**computer** Playing game and computer is calculating its move.

**end** Game is over (all coins taken).


- Only valid event is “s” to start a new game.

Listing the states is useful but we find diagrams easier to follow so would take the above information and build a diagram like the following



**Figure 2** – State and event diagram of game — “what to listen out for and when”.

The above diagram is missing some details — which we will discuss soon — but we can start coding our game based on this now.

 Save your `TakeAway_Graphical_Events.py` as `TakeAway_Graphical_States.py` and modify it so that it matches the following code.

#### TakeAway\_Graphical\_States.py

```
1 import turtle
2
3 screen = turtle.Screen()
4 bob = turtle.Turtle()
5 bob.state = "start"
6
7 def startGame():
8     print ("Starting game.")
9     print ("Do you want to play first (Y/N) ?")
10    bob.state = "start"
11
12 def player_No():
13     if bob.state == "start":
14         print("Great! I will play first ...")
15         bob.state = "computer"
16
17 def player_Yes():
18     if bob.state == "start":
19         print("OK, you can play first ...")
20         bob.state = "human"
21
22 def player_1():
23     print ("player_1 not done yet!")
24 def player_2():
25     print ("player_2 not done yet!")
26 def player_3():
27     print ("player_3 not done yet!")
28
29 screen.onkey(startGame, "s")
30 screen.onkey(player_1, "1")
31 screen.onkey(player_2, "2")
32 screen.onkey(player_3, "3")
33 screen.onkey(player_No, "n")
34 screen.onkey(player_Yes, "y")
35
36 screen.listen()
```

So what does the above code do?

- In line 5, we store the state in our turtle **bob**.
- We can press "s" at any time to start a new game, then state switches to **start**.
- When **and only when** we are in **start** state we can press "y" or "n" to select going first or second.
- When in state **human** we can press "1", "2" or "3" to make a move.

Next we need to start filling in the logic within each state...

## 2.1 Implementing State: start

When a player presses “s” to start a new game what should happen?

1. Pick a random number of coins, say in range 15 ... 21, to start game.  
We will store the number of coins in our turtle bob.
2. Output message, saying number of coins.
3. Switch state to **start** and wait for events “n” and “y”.

 Save your `TakeAway_Graphical_States.py` as `TakeAway_Graphical_Basic.py` and modify it as follows.

We want to generate random numbers so we import the `random` library, by inserting line

```
2 import random
```

Change function `startGame` to

```
8 def startGame():
9     bob.coins = random.randint(15,21)
10    print ("Starting game with %s coins." % bob.coins)
11    print ("Do you want to play first (Y/N) ?")
12    bob.state = "start"
```



At this point if you press “s” to start a new game, then press you should get a message sayin the initial number of coins.

## 2.2 Implementing State: human

We have three events to deal with — “1”, “2” and “3” — and their linked functions. But all three events will have similar behaviour so to avoid duplicate code we write a single function that the three linked functions point to

 Modify your `TakeAway_Graphical_Basic.py` to:

```
25 def player_1():
26     playerMove(1)
27 def player_2():
28     playerMove(2)
29 def player_3():
30     playerMove(3)
31
32 def playerMove(move):
33     if bob.state != "human": return
34
35     bob.coins = bob.coins - move
36     print("You have taken %s coins" % move)
37     bob.state = "computer"
```



At this point if you press “s” to start a new game, then press “y” to move first, then you can pick “1”, “2”, or “3” to play.

## 2.3 Implementing State: computer

In earlier worksheets we have covered the logic to get the computer to

- play as a naive player — just taking a random number of coins.
- playing with optimal strategy
- playing with a skill level, that can be set to any values between zero (for naive player) up to one (for optimal player).

So here we will start with the naive player and leave that more fancy strategies as an exercise.



Insert the following function into your `TakeAway_Graphical_Basic.py`

```
40 def computerMove():
41     move = random.randint(1, min(1, bob.coins))
42     bob.coins = bob.coins - move
43     print("I'm going to take %s coins" % move)
44     bob.state = "human"
```

We can't test this function yet because we are missing a **glue** function that monitors the overall state of the game

## 2.4 Linking the stages — functions update



Insert the following function into your `TakeAway_Graphical_Basic.py`

```
47 def update():
48
49     print ("Game has %s coins left." % bob.coins)
50
51     # check for game over
52     if bob.coins<=0:
53         if bob.state == "computer":
54             print("OK, OK you won")
55         else:
56             print("Look I won again!")
57         bob.state = "end"
58         drawScreen()
59         return
60
61     if bob.state == "computer":
62         computerMove()
63
64     drawScreen()
```

And at bottom of functions `startGame`, `humanMove`, `computerMove`, `player_No`, and `player_Yes` append line

```
13     update()
```

## 2.5 Implementing Graphics — function drawScreen

Currently our program won't run since we call function `drawScreen` which we have not yet defined.

 Fix that issue by inserting the function

```
66 def drawScreen():  
67     print("drawScreen not implemented yet")
```



At this point if you press "s" to start a new game, you can play the game. There is no graphics and all message are displayed at the console.

 Now to build up our graphics we replace the above function with the following

```
66     drawScreen()  
67  
68 def drawScreen():  
69  
70     print (bob.state)  
71  
72     bob.reset()  
73     bob.penup()  
74     bob.speed("fastest")  
75  
76     # title  
77     bob.goto(0,200)  
78     bob.write("Take Away", align="center", font=("Arial", 60))  
79  
80     # coins  
81     bob.goto(0,0)  
82     bob.write(bob.coins, align="center", font=("Arial", 100))  
83     bob.goto(0,-40)  
84     bob.pendown()  
85     bob.width(5)  
86     bob.circle(100)  
87  
88     # coins  
89     bob.penup()  
90     bob.goto(-300,-200)
```

Also to get the messages displayed on the graphical screen, instead of the console, we need to make the following changes:

 In function `startGame` change line

```
10     print ("Starting game with %s coins." % bob.coins)
```

to

```
10     bob.messages = ["Starting game with %s coins." % bob.coins]
```





☞ In the rest of program change any **print** line like

```
11 print ("Do you want to play first (Y/N) ?")
```

to

```
11 bob.messages.append("Do you want to play first (Y/N) ?")
```

You will need to do this in eight places!



Test!

Finally we have a finished product. Test, Test and Test!

### 3 Exercises (or no rest for the wicked)

The current game is pretty good — even if I say so myself — but it has issues. Your job is to fix the problems and make me look good!

1. You have to press “s” to start playing the game. Fix this by inserting code

76 `startGame()`

just before line

77 `screen.listen()`

2. The computer strategy is just to pick randomly. Improve on this.
3. The human player can always take up to three coins, regardless of how many coins are left. This needs to be fixed.
4. It would be nice to add a confirmation question in response to a “s” pressed event that occurs during a game, i.e., when state is **human** or **computer**.