



**AUSTRALIAN MATHS TRUST**

# Games

Workbook





# MAKING COMPUTER GAMES WITH



## **What are you going to be doing:**

This Workshop is going to be split into three parts.

You will first create a game controller using the following materials provided to you, these include:

- One MaKeyMaKey
- Playdough
- Paper
- Graphite pencil
- The others in your group, your group leader or Jess.
- Other objects in the room.

Jess will then take you through making two simple games:

- Space Invaders.
- Bubble Blaster.

It will then be your job to improve on these games and make them awesome!

## **Game Controllers: Why are they so important to games.**

Throughout the history of video games, one of the most important aspects of games is the controller. This piece of hardware is the one with which the players interact the most and is by far the most memorable component. When designed appropriately users can have an awesome and sometimes immersive experience with a computer game, in certain cases users can feel like the game controller is an extension of themselves, however if they are poorly designed can lead to users getting issues with their hands like overuse syndrome or repetitive strain injury or become frustrated and decide to not use the game console or game.

## Examples of Game controllers:

### **Dualshock Playstation game controller**



This game controller was capable of vibrating, giving feedback to the person playing the game if the game had outputs that allowed for that feature to occur.

### **Xbox Kinect**



Kinect is a motion sensing input devices that was created by Microsoft for Xbox 360 and Xbox One video game consoles and PCs. Based around a webcam-style controller, it enables users to control and interact with their console or computer without the need for a game controller, through a 'natural user interface' using gestures and spoken commands.

## Wiimote



The Wiimote, is the main for Nintendo's Wii console. A main feature of the Wii Remote is its motion sensing capability, which allows the user to interact with and manipulate items on screen via gesture recognition and pointing through the use of accelerometer and optical sensor technology.

## Joy-Con and Labo



The nintendo joy-con hey are two individual units, each containing an analog stick and an array of buttons. They can be used while attached to the main Nintendo Switch console unit, or detached and used wirelessly; when detached, a pair of Joy-Con can be used by a single player, or divided between two as individual controllers. Nintendo Labo uses kits that include cardboard cut-outs and other materials that are to be assembled in combination with the Nintendo Switch console display and Joy-Con controllers to create something called a "Toy-Con" that can interact with game software and vice versa. Nintendo designed Labo as a way to teach principles of engineering, physics, and basic programming.

### **Now it's your turn to make a game controller using the MaKeyMaKey.**

MaKey MaKey is an invention kit that connects any object that conducts even a little bit of electricity to make things like banana pianos.

With MaKey MaKey projects, you use metal alligator clips connected to the MaKey MaKey board to generate electric impulses which any computer device accepts as input. So you can browse web pages or play online games.

[Have a look at this video](#) to learn more about the MaKeyMaKey. Plus this [fun video](#) on the MaKeyMaKey.

You will be handed a makey makey and Jess will show you a video showing all the awesome things you can do with a MaKeyMaKey. When it is time open up the box, when instructed take out the MaKeyMaKey as instructed, these are fragile objects and are sensitive to electrical current. You will now do the following task:

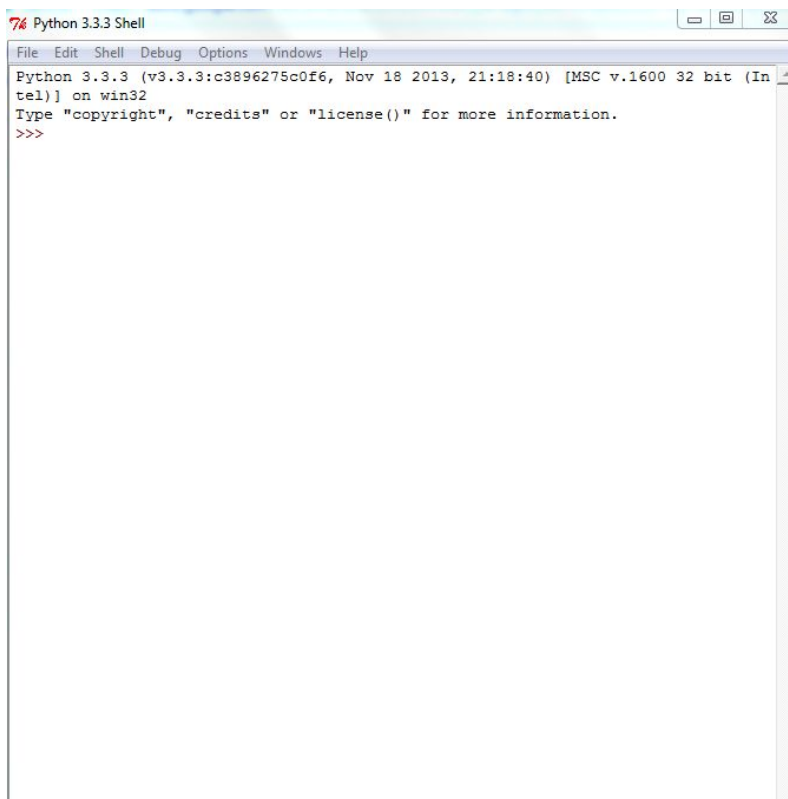
1. Connect the alligator clips up to ground, the up, down left and right arrows. Jess will instruct you on how to do this.
2. Create your Game Controller using the materials provided to you! Be as imaginative as you like!
3. We will then take the MaKeyMaKey out to the computer lab with your game controller.
4. Jess will show you how to attach the MaKeyMaKey to the computer.
5. Jess will show you how to open Python IDLE. While holding your ground control, hold the alligator clip that is attached to the right arrow on the MaKeyMaKey, when your right controller is tapped check that your cursor goes right, if you want to be extra sure the MaKeyMaKey is working do the same thing with the left.

## Creating your first game: Bubble Blaster

Bubble Blaster requires the following to happen in this game:

- The player controls a submarine.
- The arrow keys move the submarine.
- Popping bubbles scores points.
- A timer is set to 30 seconds at the start.
- Scoring 1,000 points earns extra time.
- The game ends when time runs out.

**Step 1: Create a new file, and save it in IDLE as Bubble blaster.**



```
Python 3.3.3 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

**Step 2a:** You will now create the **Graphical User Interface (GUI)** using the Tkinter Library/Module. Type out the following code.

```
#importing the Tkinter Library and all of the Tkinter functions
from tkinter import *
#setting the size of the window
HEIGHT = 500
WIDTH = 800
window =Tk ()
#Title of the game
window.title ('Bubble Blaster')
#This creates the background of the game
c = Canvas(window, width=WIDTH, height=HEIGHT, bg='darkblue')
c.pack()
```

There are a few things you need to note in this code.

- The sentence that is using the hashtag before it: This is called commenting. This is code the computer will not read while the program is running. You can choose to add it in or not.
- The words in capital letters: These are called constants. A constant is a variable that you do not want to change throughout your program. The computer in this case will not worry about the 'casing' of your letters, this is just a programming concept that allows you to tell the difference between a regular variable and a constant variable.
- You may have noticed a line at the start importing something called Tkinter. This is a Library, or some people call this a module. This is another module that can be used to create GUI in python.

At this point you can run the program. What happens when you run it?

**Step 2b:** You will create your ship. Leaving a line type out the following code directly underneath the first:

```
#creating our ship
ship_id = c.create_polygon (5, 5, 5, 25, 30, 15, fill='red')
ship_id2 = c.create_oval (0 ,0 , 30 , 30 ,outline='red')
SHIP_R = 15
#This code indicates to the computer where the middle of the screen is.
MID_X =WIDTH / 2
MID_Y=HEIGHT /2
#This tells the computer where to move the ship to
c.move (ship_id, MID_X, MID_Y)
c.move (ship_id2, MID_X, MID_Y)
```

Save your code again and run your program.



**Step 3:** You will now code your ship to move. Like before ( leaving a line in between) type out the following code directly underneath the first:

```
# allowing the ship to move
SHIP_SPD = 10
def move_ship (event):
    if event.keysym == 'Up':
        c.move(ship_id, 0, -SHIP_SPD)
        c.move (ship_id2, 0, -SHIP_SPD)
    elif event.keysym == 'Down':
        c.move(ship_id, 0, SHIP_SPD)
        c.move(ship_id2, 0, SHIP_SPD)
    elif event.keysym == 'Left':
        c.move (ship_id, -SHIP_SPD, 0)
        c.move (ship_id2, -SHIP_SPD, 0)
    elif event.keysym == 'Right':
        c.move (ship_id, SHIP_SPD, 0)
        c.move (ship_id2, SHIP_SPD, 0)
c.bind_all ( '<Key>' , move_ship)
```

At this point reconnect your MaKeyMaKey and see what the game does when using the MaKeyMaKey.

By this step your code should look like this:

```
#importing the Tkinter Library and all of the Tkinter functions
from tkinter import *
#setting the size of the window
HEIGHT = 500
WIDTH = 800
window =Tk ()
#Title of the game
window.title ('Bubble Blaster')
#This creates the background of the game
c = Canvas(window, width=WIDTH, height=HEIGHT, bg='darkblue')
c.pack()

#creating our ship
ship_id = c.create_polygon (5, 5, 5, 25, 30, 15, fill='red')
ship_id2 = c.create_oval (0 ,0 , 30 , 30 ,outline='red')
SHIP_R = 15
#This code indicates to the computer where the middle of the screen is.
MID_X =WIDTH / 2
MID_Y=HEIGHT /2
#This tells the computer where to move the ship to
c.move (ship_id, MID_X, MID_Y)
c.move (ship_id2, MID_X, MID_Y)

# allowing the ship to move
SHIP_SPD = 10
def move_ship (event):
    if event.keysym == 'Up':
        c.move(ship_id, 0, -SHIP_SPD)
        c.move (ship_id2, 0, -SHIP_SPD)
    elif event.keysym == 'Down':
        c.move(ship_id, 0, SHIP_SPD)
        c.move (ship_id2, 0, SHIP_SPD)
    elif event.keysym == 'Left':
        c.move (ship_id, -SHIP_SPD, 0)
        c.move (ship_id2, -SHIP_SPD, 0)
    elif event.keysym == 'Right':
        c.move (ship_id, SHIP_SPD, 0)
        c.move (ship_id2, SHIP_SPD, 0)
c.bind_all ( '<Key>' , move_ship)
```

**Step 4:** You are now going to create your bubbles. For now you won't be able to play your game while you are setting your bubbles up. Type out the following code:

```
#importing the radom Library and only the random integer functions.
from random import randint
#These are lists
#stores the ID number of the bubble so the program can move it later.
bub_id = list ()
#stores the radius of the bubble
bub_r = list ()
#stores how fast the bubble travels across the screen
bub_speed = list()
MIN_BUB_R = 10
MAX_BUB_R = 30
MAX_BUB_SPD = 10
GAP = 100
```

**Step 5:** Underneath this code type out the following:

```
# creating the bubbles
def create_bubble():
    x = WIDTH + GAP
    y = randint(0, HEIGHT)
    r = randint(MIN_BUB_R, MAX_BUB_R)
    id1 = c.create_oval(x - r, y - r, x + r, y + r, outline='white')
    bub_id.append(id1)
    bub_r.append(r)
    bub_speed.append(randint(1, MAX_BUB_SPD))

# moving the bubbles
def move_bubbles():
    for i in range(len(bub_id)):
        c.move(bub_id[i], -bub_speed[i], 0)

# finding out where a particular bubble is.
def get_coords(id_num):
    pos = c.coords(id_num)
    x = (pos[0] + pos[2]) / 2
    y = (pos[1] + pos[3]) / 2
    return x, y

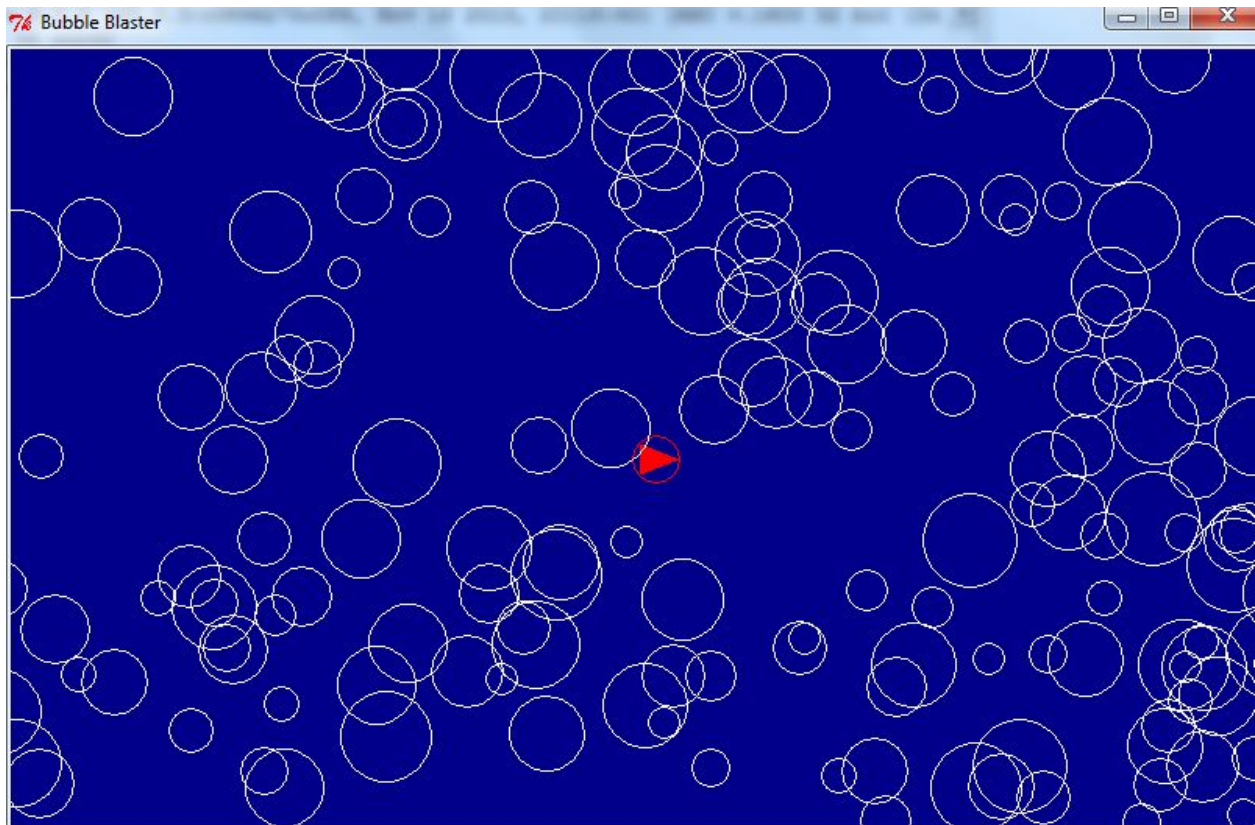
# removing a bubble from the game
def del_bubble(i):
    del bub_r[i]
    del bub_speed[i]
    c.delete(bub_id[i])
    del bub_id[i]

# cleaning up a bubble that has floated off the screen
def clean_up_bubs():
    for i in range(len(bub_id) - 1, -1, -1):
        x, y = get_coords(bub_id[i])
        if x < -GAP:
            del_bubble(i)
```

**Step 6:** Now we will create the start of the main game **loop**, we will come back to it later to add more lines of code. However to start it off add the following. You can now run your game again.

```
from time import sleep, time
BUB_CHANCE = 10
# Main loop for the game.
while True:
    if randint(1, BUB_CHANCE) == 1:
        create_bubble()
        move_bubbles()
        clean_up_bubs()
        window.update()
        sleep(0.01)
```

Your game should look similar to this now.



If it doesn't let someone know. There could be one of three issues. These are called **Syntax**, **Logic** and **Runtime** errors. A common error that occurs with Jess is that she finds she indents her code wrong or leaves out a bracket. It does get frustrating!

**Step 7:** In many games it is important to know the distance between two objects. We are going to get the computer using the following **algorithm** to figure out this distance. Type the following code directly after the `clean_up_bubs` section. At this stage you can still run the game however you will notice not much has changed, If the game still runs this means your game is ok!

```
from math import sqrt
def distance (id1,id2):
    x1, y1 = get_coords(id1)
    x2, y2 = get_coords(id2)
    return sqrt ((x2 -x1) **2 + (y2-y1) **2)

def collision():
    points=0
    for bub in range (len(bub_id)-1,-1,-1):
        if distance(ship_id2,bub_id[bub])<(SHIP_R+bub_r[bub]):
            points += (bub_r[bub] + bub_speed[bub])
            print("boom")
            del_bubble(bub)
    return points
```

**Step 8:** This section of code allows the the computer to work out when each bubble is popped by using the radius of each bubble. Add this code directly after the step 7 code. Run the game again to see if any errors come up. The game won't fully work yet, but it will soon!

```
def collision ():
    #points variable keeps track of the score
    points = 0
    for bub in range (len(bub_id) -1, -1, -1):
    #checks for collisons between the submarine and any bubbles.
        if distance (ship_id2, bub_id [bub]) < (SHIP_R + bub_r [bub]):
    #this calculates the bubbles and adds it to the points
            points += (bub_r[bub] + bub_speed [bub])
            del_bubble (bub)
    #returns the number of points back to the player
    return points
```

**Step 9:** Now that you have added your pop the bubble code, you need to add an extra few lines to the main loop for the game. This is important code, you may not see a difference yet when played but you will see the difference in the next step.

```
from time import sleep, time
BUB_CHANCE = 10
# Main loop for the game.
score = 0
while True:
    if randint (1, BUB_CHANCE) == 1:
        create_bubble()
        move_bubbles()
        clean_up_bubs()
        score += collision()
        print (score)
        window.update ()
        sleep (0.01)
```

The main changes are adding score = 0 in between BUB\_CHANCE and score+ collision () and print (score) in between clean\_up\_bubs and window.update.

**Step 10:** You will now create your score and time labels for your game. After your def collision () section type the following code:

```
# creating the time label
c.create_text (50, 30, text='TIME' , fill='white' )
#creating the score label
c.create_text (150, 30, text='SCORE' , fill='white' )
#making this appear in the game
time_text = c.create_text ( 50, 50, fill= 'white')
score_text = c.create_text (150, 50, fill= 'white')
def show_score(score):
    c.itemconfig (score_text, text=str(score))
def show_time(time_left):
    c.itemconfig(time_text, text=str(time_left))
```

**Step 11:** the final touches! We are nearly there! Add the following code to the start of the main game loop.

```
from time import sleep, time
BUB_CHANCE = 10
TIME_LIMIT = 30
BONUS_SCORE =1000
# Main loop for the game.
score = 0
bonus = 0
end = time() + TIME_LIMIT
```

**Step 12:** You will need to update the main game loop to include some new code. All of the main game loop has been included this time so you can make sure that your game has all of the final code to make the game work.

```

from time import sleep, time
BUB_CHANCE = 10
TIME_LIMIT = 30
BONUS_SCORE = 1000
# Main loop for the game.
score = 0
bonus = 0
end = time() + TIME_LIMIT
while time () < end:
    if randint (1, BUB_CHANCE) == 1:
        create_bubble()
        move_bubbles()
        clean_up_bubs()
        score += collision()
    if (int(score / BONUS_SCORE)) > bonus:
        bonus += 1
        end += TIME_LIMIT
    show_score (score)
    show_time (int(end -time ()))
    window.update ()
    sleep (0.01)

```

**Step 13:** Now it is time to create a GAME OVER graphic. After the main game loop add the following code:

```

c.create_text (MID_X, MID_Y, text = 'GAME OVER', fill= 'white',
               font =('Helvetica', 30))
c.create_text (MID_X, MID_Y +30, TEXT = 'Score:' + str (score), fill='white')
c.create_text (MID_X, MID_Y +45,
               text= 'Bonus time:' +sr (bonus*TIME_LIMIT), fill= 'white')

```

**Step 14:** Now it is time to play your game with the MaKeyMaKey!

Here are some suggestions on how to improve your game:

- Make the game harder: by adjusting the time limit, the score for bonus time and the amount of bubbles the submarine can pop.
- Choose a different colour for your background and for the submarine.
- Add a smart bomb that deletes all of the bubbles when you press the spacebar.
- Build a leaderboard to keep track of the best scores

## Your second game: Space invaders.

The game we are about to create is a tribute to the original space invaders game created by Tomohiro Nishikado. It is considered a very influential game in game design history as many game creators, who for example created games like the Legend of Zelda and Donkey Kong believe it was the first game they came into contact with.

You may learn in Jan's workshop that the first program you will write is called Hello World. In game design your first game usually is a version of space invaders!

In game design you will be learning a important skill of creating games with a Graphical User Interface (GUI). This workshop will help you to create projects with graphics using special Libraries/modules that allow Python to have GUI's.

A Python library is a collection of functions and methods that allows you to perform lots of actions without writing your own code.

In Space invaders you will be using a Library called Turtle. Turtle was a library developed by Wally Feurzig and Seymour Papert in 1966/67 to teach programming concepts to beginner programmers, which make programming visually pleasing. Developers of python decided to put the turtle Library in python for the exact same reason. You will also be using two extra modules called random and math. What do you think they do?

### Time to create Space invaders:

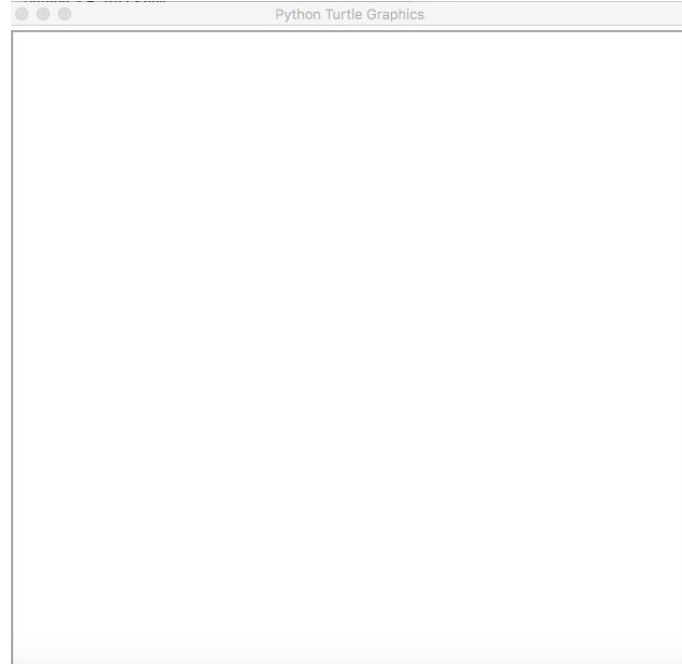
**Step one:** import your modules and create your window, type out the following:

```
#importing turtle, math and random Libraries
import turtle
import math
import random

#window creates a screen object called window
window = turtle.Screen()
window.bgcolor("white")
```



Now run your game. The following window should appear!



**Step 2:** Create your border, scoring system. Type the following code in:

```
#the for loop is there to tell the computer to tell the border pen to outline the window
for side in range(4):
    border_pen.fd(600)
    border_pen.lt(90)
    border_pen.hideturtle()

score = 0

score_pen = turtle.Turtle()
score_pen.speed(0)
score_pen.color("black")
score_pen.penup()
score_pen.setposition(-290, 280)
scorestring = "Score: %s" % score
score_pen.write(scorestring, False, align="left", font=("Arial", 14, "normal"))
score_pen.hideturtle()
```

So far you should have the following code ( hint don't type it out again):

```
#importing turtle, math and random Libraries
import turtle
import math
import random

#window creates a screen object called window
window = turtle.Screen()
window.bgcolor("white")

#creates a turtle object called border_pen which creates a border around the window
border_pen = turtle.Turtle()
border_pen.speed(0)
border_pen.color("white")
border_pen.penup()
border_pen.setposition(-300, -300)
border_pen.pendown()
border_pen.pensize(3)

#the for loop is there to tell the computer to tell the border pen to outline the window
for side in range(4):
    border_pen.fd(600)
    border_pen.lt(90)
    border_pen.hideturtle()

score = 0

score_pen = turtle.Turtle()
score_pen.speed(0)
score_pen.color("black")
score_pen.penup()
score_pen.setposition(-290, 280)
scorestring = "Score: %s" % score
score_pen.write(scorestring, False, align="left", font=("Arial", 14, "normal"))
score_pen.hideturtle()
```

If you have all of that code you can run the game to see if it is working. The main difference you should see in the screen is the score.

**Step 3:** Create your player object. Code the following:

```
#this creates a player object using the turtle pen
player = turtle.Turtle()
player.color("blue")
player.shape("triangle")
player.penup()
player.speed(0)
player.setposition(0, -250)
player.setheading(90)
#player speed
playerspeed = 15
```

By this point you should have a 'player' at the bottom of the screen but it shouldn't move.

#### Step 4: Create your enemies:

```
#creating a list of enemies
number_of_enemies = 5
enemies = [] #this is another way to make a list

#This adds more enemies to the list
for i in range(number_of_enemies):
    enemies.append(turtle.Turtle())

for enemy in enemies:
    enemy.color("red")
    enemy.shape("circle")
    enemy.penup()
    enemy.speed(0)
    x = random.randint(-200, 200)
    y = random.randint(100, 250)
    enemy.setposition(x, y)
#enemy speed
enemyspeed = 2
```

This will create your enemies or your 'space invaders.' like before, they will appear but they will not move.

#### Step 5: Create your bullet.

```
#this creates the bullet
bullet = turtle.Turtle()
bullet.color("yellow")
bullet.shape("triangle")
bullet.penup()
bullet.speed(0)
bullet.setheading(90)
bullet.shapesize(0.5, 0.5)
bullet.hideturtle()

#bullet speed
bulletspeed = 10

bulletstate = "ready"
```

Now you are ready to make the functions that will allow you to move your objects in your game:

**Step 6:** Create the functions that will make your objects move. You will need to wait to step 8 to see how this takes effect.

```
#def define new function
def move_left():
    x = player.xcor()
    x -= playerspeed
    if x < -280:
        x = - 280
    player.setx(x)

def move_right():
    x = player.xcor()
    x += playerspeed
    if x > 280:
        x = 280
    player.setx(x)

def fire_bullet():
    global bulletstate
    if bulletstate == "ready":
        bulletstate = "fire"
        x = player.xcor()
        y = player.ycor() + 10
        bullet.setposition(x, y)
        bullet.showturtle()

def isCollision(t1, t2):
    distance = math.sqrt(math.pow(t1.xcor() - t2.xcor(), 2) + math.pow(t1.ycor() - t2.ycor(), 2))
    if distance < 15:
        return True
    else:
        return False
```

**Step 7:** Create your key bindings. This allows for you to start making your game interactive!

```
#key bindings
turtle.listen()
turtle.onkey(move_left, "Left")
turtle.onkey(move_right, "Right")
turtle.onkey(fire_bullet, "space")
```

At this point you will be able to move the player but not the bullet.

**Step 8:** Create the main game loop.

```
#main game loop

game_state = True

while game_state:

    for enemy in enemies:
        x = enemy.xcor()
        x += enemyspeed
        enemy.setx(x)

        if enemy.xcor() > 280:
            for e in enemies:
                y = e.ycor()
                y -= 40
                e.sety(y)
                enemyspeed *= -1

        if enemy.xcor() < -280:
            for e in enemies:
                y = e.ycor()
                y -= 40
                e.sety(y)
                enemyspeed *= -1

    if isCollision(bullet, enemy):
        bullet.hideturtle()
        bulletstate = "ready"
        bullet.setposition(0, -400)
        x = random.randint(-200, 200)
        y = random.randint(100, 250)
        enemy.setposition(x, y)
        score += 1
        scorestring = "Score: %s" % score
        score_pen.clear()
        score_pen.write(scorestring, False, align="left", font=("Arial", 14, "normal"))
```

Run the program. By this time you should see the space invaders moving down the screen.

**Step 9:** Get the bullet hitting enemies and having a game over sequence by typing the following code:

```
if bulletstate == "fire":
    y = bullet.ycor()
    y += bulletspeed
    bullet.sety(y)

if bullet.ycor() > 275:
    bullet.hideturtle()
    bulletstate = "ready"

if isCollision(player, enemy):
    player.hideturtle()
    enemy.hideturtle()
    print("Game Over")
    game_state = False
```

By the end of this step if you get a fully working game. Congratulations! Now here are some challenges for you to try if you have time or for you to try at home:

Jess will show you how to research for ideas relating to improving code. After this, try to figure out some new ways to improve the code, here are some suggestions:

- Creating your own space invader graphics.
- Having a timer.
- Creating a bonus round if the player reaches a certain amount of points.
- Any other improvement you wish to see!



# ADVANCED COMPUTER GAMES WITH



EMOJIMEMORY!  
Bit.ly link: <http://bit.ly/digitemoji>

## **Welcome to Advanced Games!**

The first game you are going to create is called Emojimemory! Emojimemory is a matchmaking game, with a little twist. It uses emojis instead of simple symbols!

In this game you will be using Tkinter to display a button grid. Tkinter's mainloop function listens for button presses and handles them with a special function called a lambda.

In the general workshop you used the python keyword called def to define functions, like moving your player object right, left and to fire in your space invaders game and creating your bubbles and moving your bubbles in your bubble blaster game. As there were not many functions that you needed to create in your game, it was best in those cases to use the def function. However in this game there are cases where typing out functions may not be an efficient use of your time. This is where the Lambda function is used. Sometimes people use lambda functions in coding competitions where creating solutions quickly is the name of the game!

To make this game you will be using something called unicode to provide your emoji's. Unicode is a standard for converting most of the world's writing systems into 1's and 0's, every character is assigned a number. The English alphabet represents part of the first 128 characters of unicode. Emoji's became part of the unicode family in 2010.



## Now to create Emojimemory

1. Create a new file called emojiemory.py
2. Add the following modules to the start of your game.

```
import random
import time
from tkinter import Tk, Button, DISABLED, font
#import font too, so to make the symbols bigger
#disabled stops a button responding after a symbol is matched
```

3. Set up the GUI using the following code

```
root = Tk()
root.title("emojmemory")
root.resizable (width=False, height=False)
```













4. Test your code, you should see the following window



5. Under the code from step three add the variables that the program needs and create a dictionary to store the buttons in.

```
#This is the dictionary for the buttons
buttons = {}
#This variable is used to check if the symbol is the first in the match
first = True
#These two variables keep track of the last button pressed.
previousX = 0
previousY = 0
```

6. Add your emoji's. To choose your emoji's, you will need to find out the unicode numbers to add to your program. You will need to choose 12 emoji's. Go to <http://bit.ly/emojimem> to find the unicode characters that you want. To find out the unicode numbers, look under the code heading of the unicode website and write down the codes you wish to use. Make sure that when writing down your code you take out the plus (+) symbol and replace with a backslash (\) between the U and the number. Also make sure your u is lowercase.

<u>No</u>	<u>Code</u>	<u>Browser</u>	<u>Appl</u>	<u>Goog</u>	<u>Twtr</u>	<u>One</u>	<u>FB</u>
1	<a href="#">U+1F600</a>						
2	<a href="#">U+1F603</a>						
3	<a href="#">U+1F604</a>						
4	<a href="#">U+1F601</a>						
5	<a href="#">U+1F606</a>						
6	<a href="#">U+1F605</a>						
7	<a href="#">U+1F923</a>						

7. Type the following code below to ensure that your emoji's are included in the game.

```
#The symbol for each button is stored in this dictionary
button_symbols = {}

symbols = [ u'\u2702' , u'\u2705', u'\u2708' , u'\u2709' , u'\u270A',u'\u270B',
           u'\u270C' , u'\u270F', u'\u2712' , u'\u2714' ,u'\u2716' , u'\u2728']*2
#multiplying a list creates a single list with copies of the items
```

8. You don't want the symbols to appear in the same place every time the game is played, it would become boring for a person to play! To prevent this you need to shuffle the symbols before the game starts. Add the following line after the list of symbols.

```
random.shuffle (symbols)
```

9. The grid of buttons in your game will consist of 24 buttons arranged into four rows of six. In this next bit of code you will need to use something called a nested loop. In a nested loop, an inner loop runs inside an outer loop. In this case each time the outer loop runs (six times) the inner loop runs four times. In total the inner loop runs  $6 \times 4 = 24$  times (do you see the correlation?). Underneath the random.shuffle function write the following code:

```
for x in range (6):
    for y in range (4):
        button = Button(command=lambda x=x, y=y: show_symbol (x, y), \
                        width=5, height=5)#make the tiles bigger
#Backslash character is there as an escape character, it allows for you to
#split your code over two lines.
        button.grid(column=x, row=y)
        button.config(font=("Courier", 20))
#font Size
#this line above places the button on the GUI
        buttons [x, y] = button
#the line above saves each button in the buttons dictionary.
        button_symbols [x, y] = symbols.pop()
```

10. Start the Tkinter main loop, where the GUI will be displayed. By doing this the program will start listening for button presses. Type the following line underneath the code you wrote in step 9.

```
root.mainloop ()
```

Run your program, you should see the following appear. You can click on the buttons but they will do nothing yet.

emojimemory


11. Now for the fun bit, you are going to create the function that handles the button presses. We will not use a lambda function as it's a function that is used for button presses compared to the lambda function as this function is a little more complicated and requires a function definition rather than a lambda.

For first button presses it remembers which button is pressed, if it's the second turn it needs to check if the symbols match. Symbols that don't match are hidden. Matching symbols are left showing and their buttons are disabled.

Straight after the module declarations ( hint right at the start after the 'from tkinter' part write the following):

```
def show_symbol (x, y):
#the above function tells the function which button has been pressed.
    global first
    global previousX, previousY
#the above lines tell the program that the variables are global
    buttons [x, y] ['text'] =button_symbols [x ,y]
    buttons [x, y].update_idletasks()
#The above lines show the symbols

#If it's the first turn, the code remembers the button press by storing the
#x and y coordinates
    if first:
        previousX = x
        previousY = y
        first =False
#second turn. This line includes a check to stop the player cheating by pressing
#every button twice
    elif previousX != x or previousY != y:
        if buttons [previousX, previousY] ['text'] != buttons [x , y] ['text']:
            time.sleep (0.5)
            buttons [previousX, previousY] ['text'] = ''
            buttons [x, y] ['text'] = ''
        else:
            buttons [previousX, previousY] ['command'] =DISABLED
            buttons [x, y] ['command'] =DISABLED
    first= True
```

### How does this function work:

The function show the button's symbol by changing it's text label to the unicode character that we radomly assign to it. We use update\_idletasks() to tell Tkinter to show this symbol right now. If it's the first turn we just store the button's co-ordinates in the variables (previous x and previous y). If it's the second turn we need to check the player isn't trying to cheat by pressing the same button twice. If they aren't, we check if the symbols match. If the symbols don't match, we hide them by setting the text to empty strings ( buttons [x, y] ['text'] = ""). If they do match we leave them showing but disable the buttons ( buttons [x, y] ['command'] =DISABLED).

**How does the lambda work in this code:**

Each time the loop runs the lambda function saves the current button's row and column location. When the button is pressed, it calls the `show_symbol ()` function with these values, so the function knows which button has been pressed and which symbol to reveal.

**CONGRATULATIONS YOU HAVE A COMPLETED GAME!**  
**BUT WAIT THERE IS MORE!**

## Tweaking your game

Part of the fun of creating a game is tweaking it! Now it is your job to tweak it to make it all yours!

To help you tweak your game here are some websites that can help you:

- [Graphical User Interfaces with Python.](#)
- [Python GUI examples.](#)
- [Invent with python.](#)
- [Tkinter reference: A GUI for python.](#)

Stuck on how to tweak your game here are two suggestions:

### **Show the number of moves**

As the current game stands, the player has no way of knowing how well they have done or if they have done any better. This addition will allow you to do this:

1. Import Tkinter's message box widget to display the number of moves at the end of the game. In the import line add the word messagebox after font.

```
import random
import time
from tkinter import Tk, Button, DISABLED, font, messagebox
```

2. You will need to make new variables for this tweak. This will keep track of the number of moves the player makes, while the other remembers how many pairs the user has found. Give these variables a starting value of 0. Place these variables under the previousY variable:

```
moves = 0
pairs = 0
```

3. Make the moves and pairs variables global. They will need to be changed by the show\_symbol () function, so let that function know this by putting these two lines near the top of the function.

```
global moves
global pairs
```

4. A move is two button presses. So you will only need to add one to the moves variable when the show symbol is called for the first or second button press, so in this case we will do it for the first button press. Add the following to the show symbol function:

```
if first:
    previousX = x
    previousY = y
    first = False
    moves = moves + 1
```

5. Display a message: Add the following code near the bottom of the show\_symbol() function. It will track the matched pairs and show a message box at the end of the game telling the player how many moves they took.

```
pairs = pairs + 1
if pairs == len(buttons)/2:
    messagebox.showinfo ('Matching', 'Number of moves:'+
        str(moves) , command=close_window)
```

How does this work: There are 12 pairs of symbols, so you could have typed pairs== 12 in this tweak. However, your code is smarter than this. However., your code is smarter than this. It calculates the number of pairs by using the pairs==len (buttons)/2. This allows you to add more buttons without having to update the code.

6. You need to create a close\_window() function, to make the program exit the game when the player clicks the ok button on the Number of moves message box. Add this code under the line that imports the modules.

```
def close_window (self):
    root.destroy()
```

### Add more buttons.

Let's really challenge the player's memory by adding more and symbols to the game.

1. Add more pairs to the symbols list, choose more emoji's and include an extra line to the code.
2. Now add an extra row of buttons, change the y range in the nested loop from 4 to 5.
3. You now should have a total of 30 buttons. If you want to add more buttons make sure that gthe number of extra buttons you add is a multiple of 6, so that you always have complete rows. If you are feeling adventurous, you could experiment with different layouts by changing the nested loops.