wikiHow

How to Program Computer Games

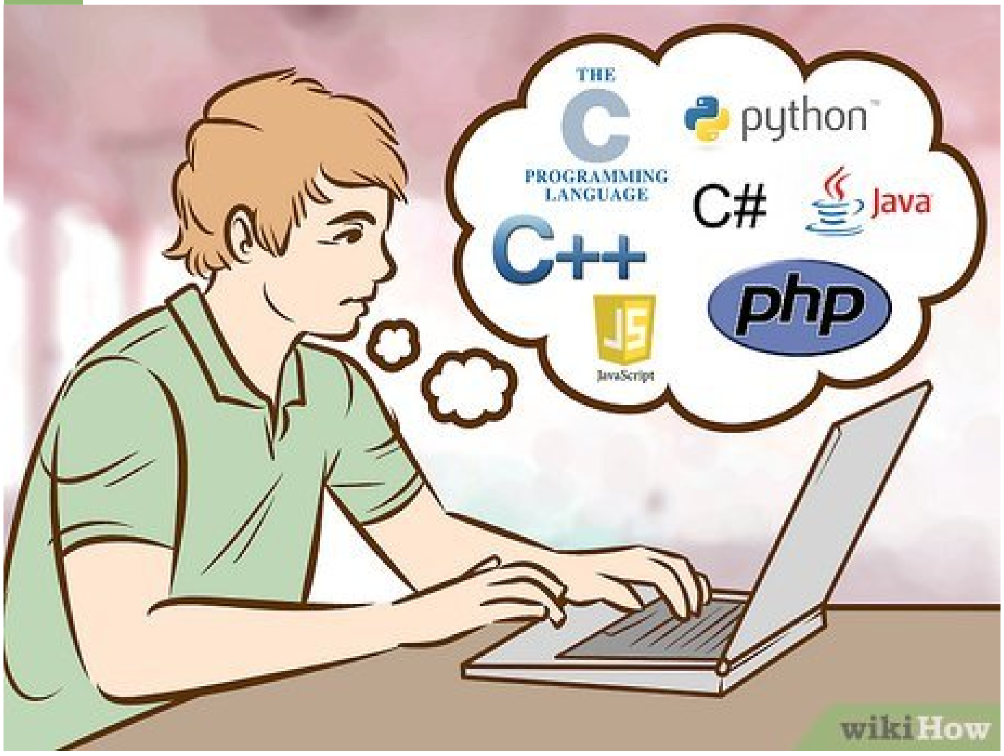
Co-authored by Nicole Levine, MFA and 18 contributors

Last Updated: June 23, 2021

Do you have an idea for a computer game and want to make it real? Or have you ever wondered how computer games are written? This wikiHow teaches you how to write three basic computer games in Python. You'll need a basic understanding of Python and general programming concepts to develop your first game.

Part
1

Part 1 of 3:
Making a Text-Based Game

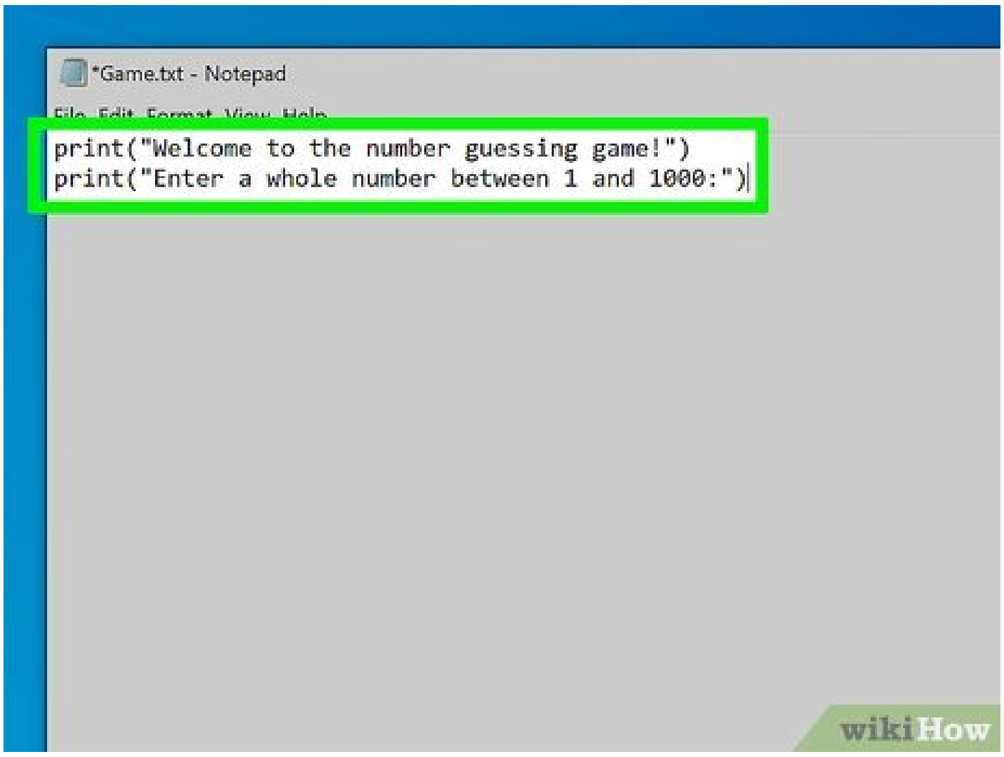


1 **Choose a programming language.** All programming languages are different, so you will have to decide which one to use to write your game. Every major programming language supports text input, text output and if-constructions (the main things you need for a simple text-based game), so explore the options and decide which you feel most comfortable with and dedicated to learning. Here are some factors to consider:

- **What is the language mostly used for?** Some programming languages, like JavaScript, are designed to be used for the web, while others, like Python, C, or C++, are designed to run computer programs. For your game, aim for a language with a broader range of use, such as Python, C, C++, or **JavaScript**.
- **How hard is it to learn?** Although writing a program should be easy enough after some practice in any normal programming language (i.e., not one specifically designed to be confusing like Malbolge), some are friendlier to beginners than others. Java and C, for example, will require you to understand deeper programming concepts than something like Python, which is known for its more accessible and straight-forward syntax.
- **Where can I use it?** You probably want people on different systems, such as Linux, Mac or Windows, to all be able to play your game. So you shouldn't use a language that is only supported on a few systems, like Visual Basic, which is only supported on Windows.

This article will use Python for the examples of a text-based game, but you can look up how the concepts are done in any other programming language.

2 **Get your computer ready.** The two main components you'll need are a text editor, in which you'll write your code, and a compiler, which you'll use to turn it into a game. If you want to follow the example in this article, you should install Python and learn how to run programs. If you want to, you can set up an IDE (Integraded Desktop Environment), which combines editing, compiling, and debugging into a single program. Python's IDE is called IDLE. But you can also just use any text editor that supports plain text, such as Notepad for Windows, TextEdit for macOS, or Vim for Linux.

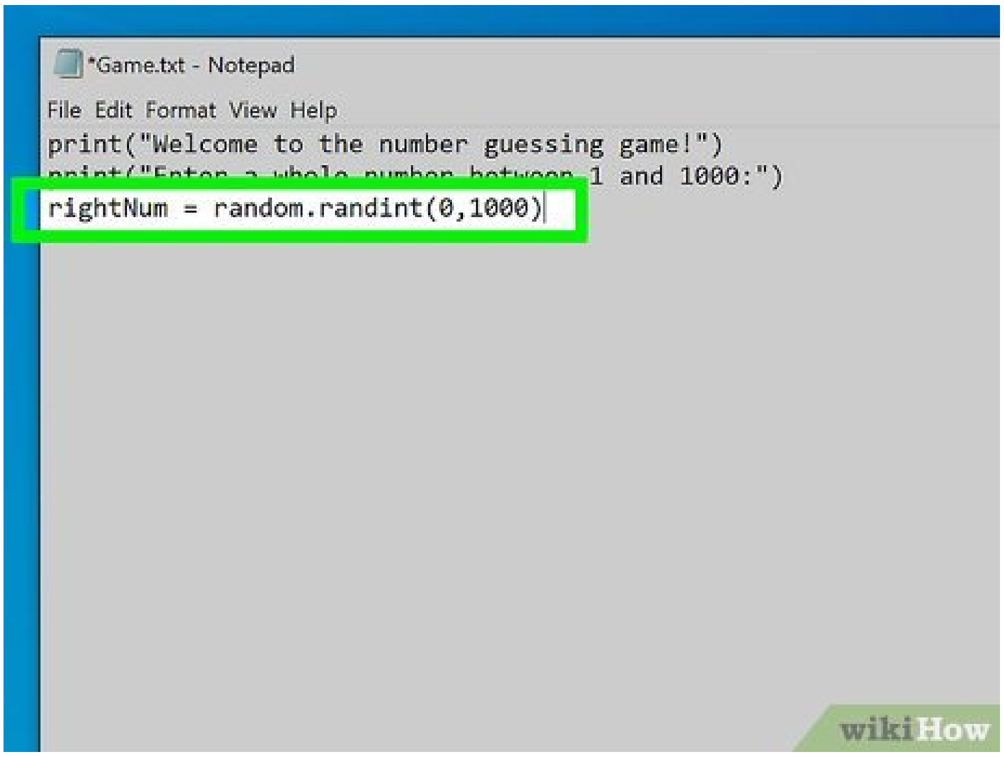


```
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
```

3 **Write some code to greet the player.** The player will want to know what is going on and what they have to do, so you should print some text for them.

- This is done with the `print()` function in Python. To try it out, open a new file with the .py extension, enter the following code into it, save and run it:

```
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
```

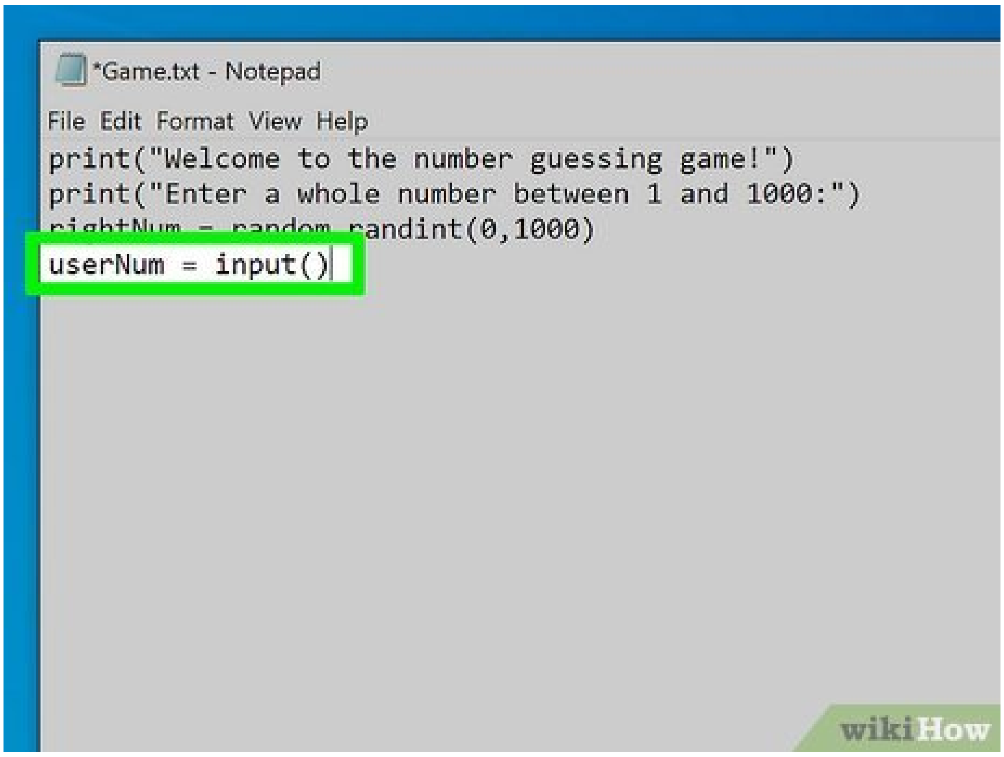


```
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
```

4 **Generate a random number.** Let's make a text-based game that asks the player to guess the correct number. The first thing we'll need to do is generate a random number at the start of the game so the player won't always guess the same number. Since the number will remain the same throughout the program, you'll want to store the random number in a variable.

- Python doesn't have a built-in random number function, but it does have a standard library (this means the user won't have to install anything extra) that does. So go to the beginning of your code (before the `print()` functions) and type the line `import random`.
- Use the random function. It is called `randint()`, is in the `random` library which you just imported, and takes the minimal and maximal value the number can have as argument. So go back to the end of your code and enter following line:

```
rightNum = random.randint(0,1000)
```

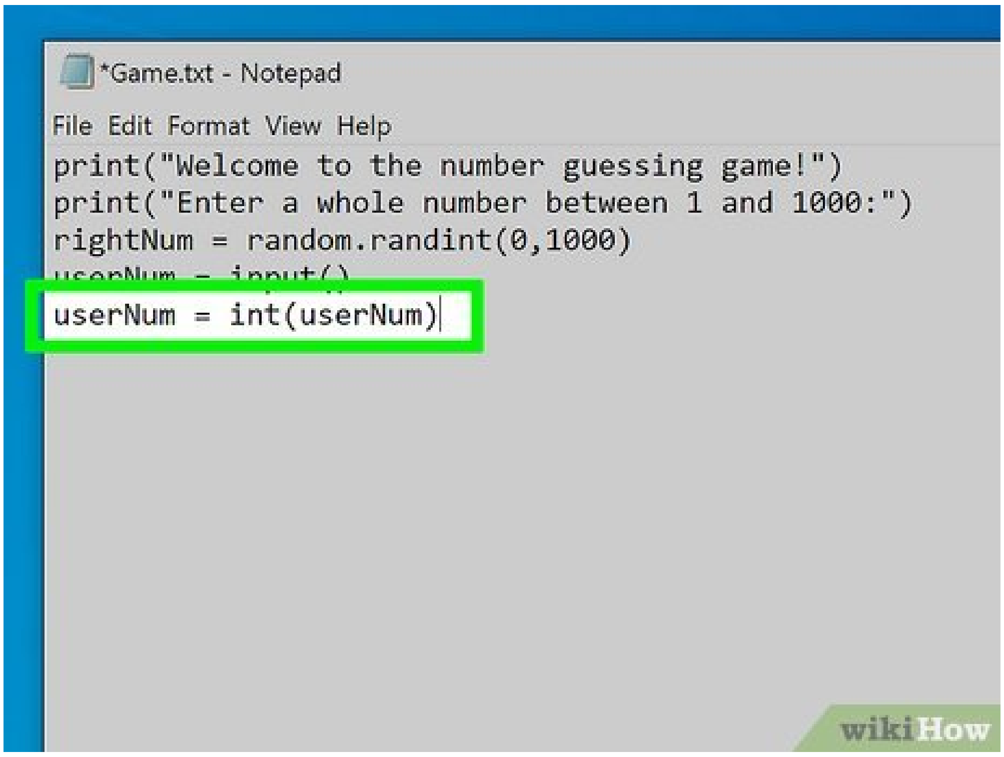


```
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
```

5 **Get input from the player.** In a game, the player wants to do something or interact with something. In a text-based game, this is possible by entering text. Now that we have a random number, our next lines of code should ask the player to input their best guess.

- Since the code you entered prints the instruction to enter a number to the player, it should also read the number they enter. This is done with `input()` in Python 3, and `raw_input()`in Python 2. You should write in Python 3, as Python 2 will become outdated soon. Add the following line to your code to store the player's input in a variable called number:

```
userNum = input()
```

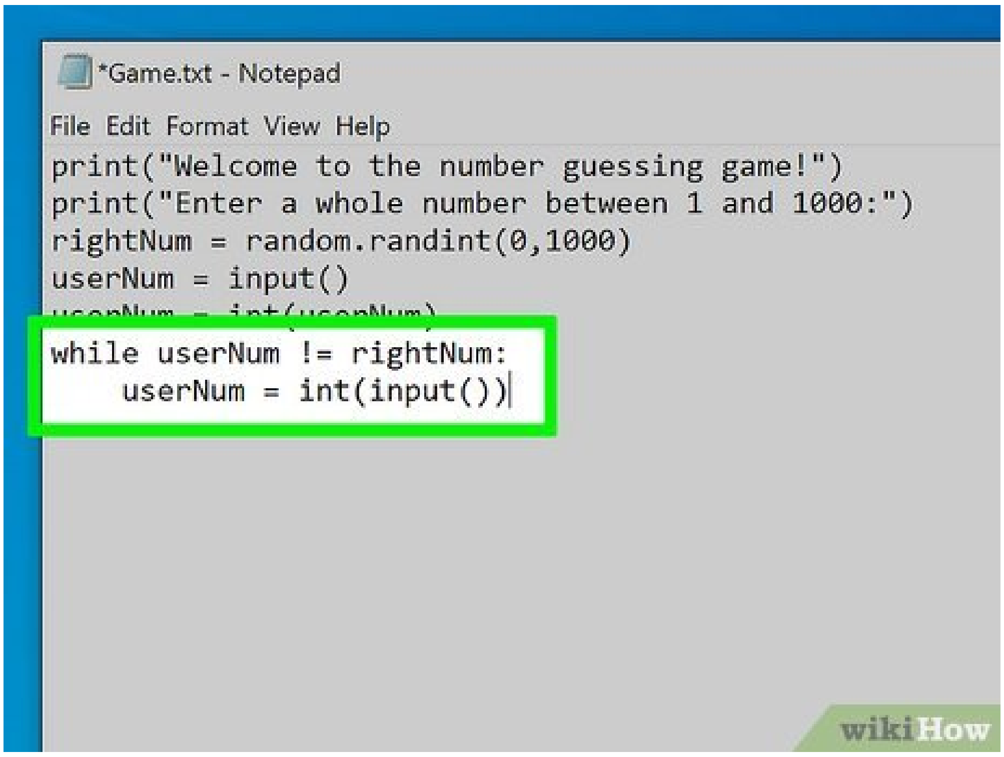


```
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
userNum = int(userNum)
```

6 **Turn the player's input into a usable data type.** The player has entered a number—now what?

- Make the player's input a number. Now, this might sound confusing because they just entered a number. But there is a good reason: Python assumes that all input is text, or a "string," as it's called in programming. This text contains the number you want to get. Python has a function that converts a string that only contains a number to the number inside. Type:

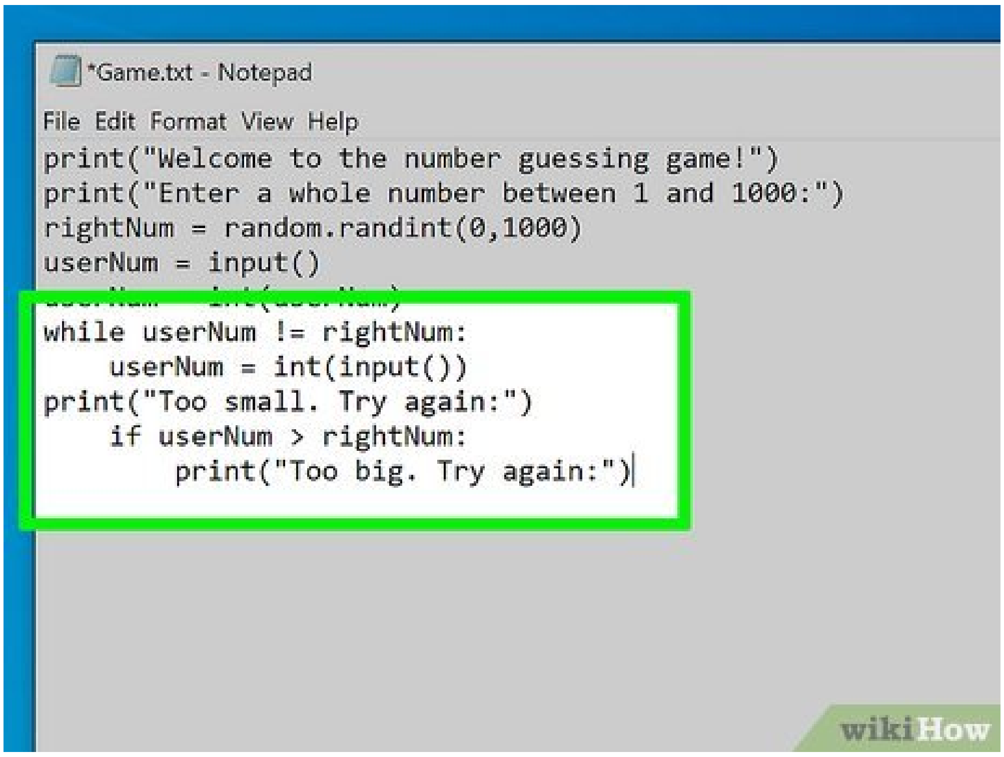
```
userNum = int(userNum)
```



```
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
userNum = int(userNum)
while userNum != rightNum:
    userNum = int(input())
```

7 **Compare the player's number to the correct number.** Once the player inputs their number, you'll need to compare it to the one that was randomly generated. If the numbers aren't the same, your game can make the player try another number. If the numbers match, you may tell the player they guessed correctly, and quit the program. This is done with the following code:

```
while userNum != rightNum:
    userNum = int(input())
```

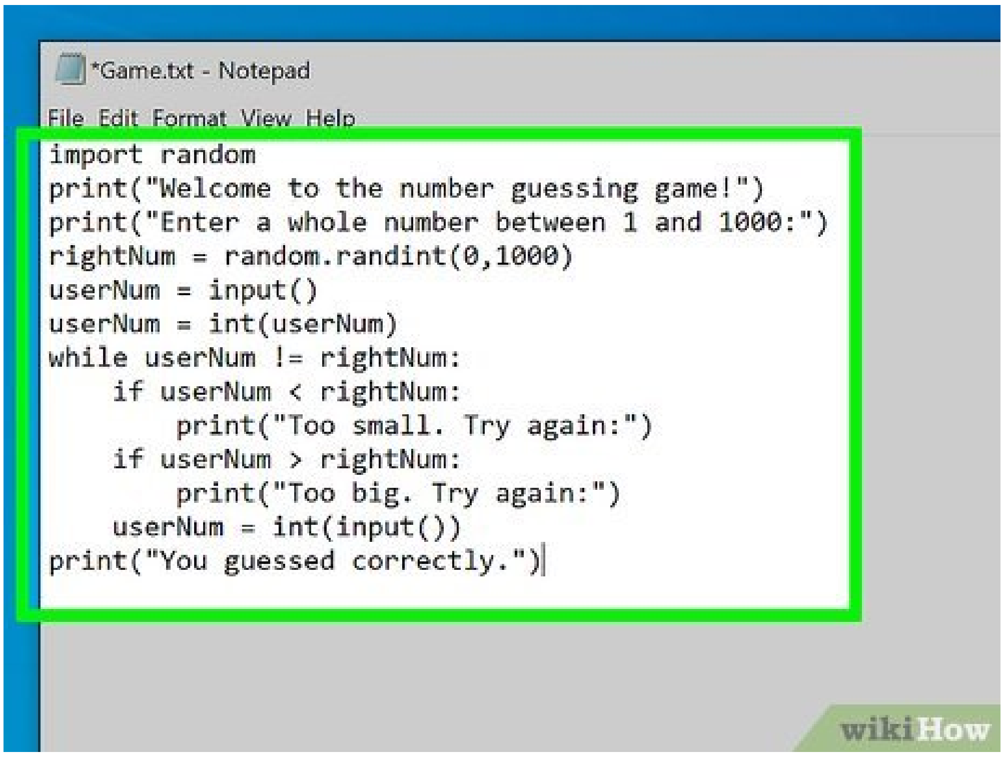


```
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
userNum = int(userNum)
while userNum != rightNum:
    userNum = int(input())
print("Too small. Try again:")
    if userNum > rightNum:
        print("Too big. Try again:")
```

8 **Give the player feedback.** While you already have processed their input, the player won't see this. You'll need to actually print the results to the player so they understand what's happening.

- Surely, you could just tell the player whether their number is right or wrong. But with that approach, the player might have to guess 1000 times in the worst case, which would be very boring.
- So tell the player whether their number is too small or too big. This will reduce their number of guesses significantly. If, for example, the player guesses 500 first, and the game replies "Too big. Try again," there will only be 500 possible numbers instead of 1000. This is done with if-constructions, so replace the `print("Wrong. Try again.")` with one.
- Be aware that checking whether two numbers are the same is done with == , not with = . = assigns the value right of it to the variable left of it!

```
if userNum < rightNum:
    print("Too small. Try again:")
if userNum > rightNum:
    print("Too big. Try again:")
```

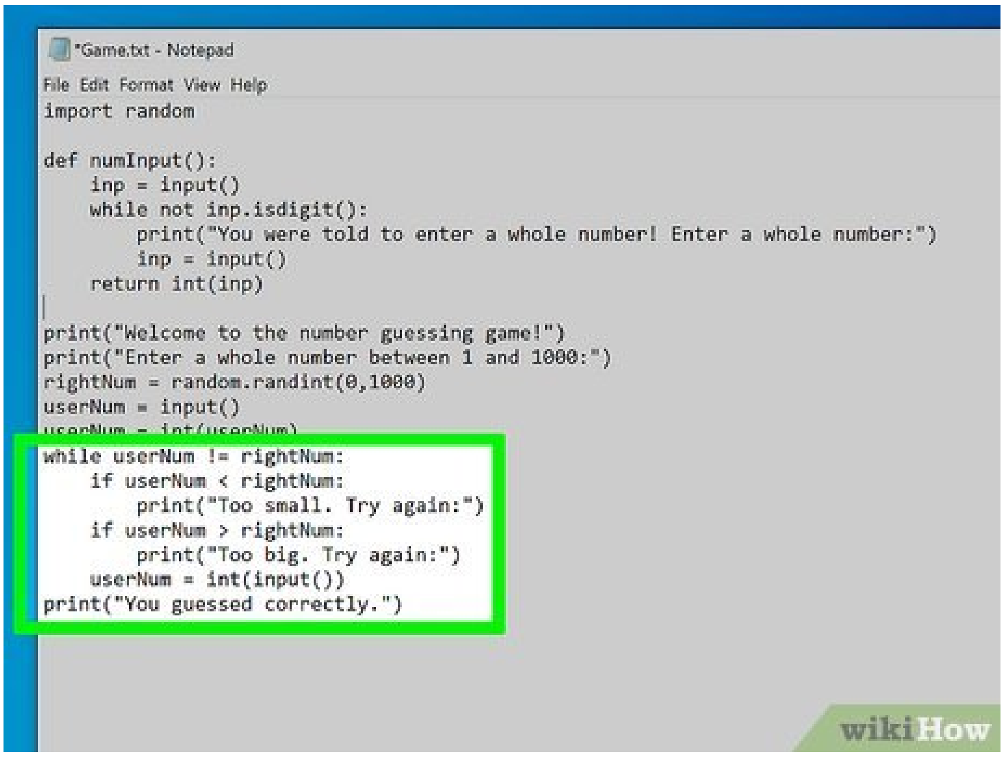


```
import random
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
userNum = int(userNum)
while userNum != rightNum:
    if userNum < rightNum:
        print("Too small. Try again:")
    if userNum > rightNum:
        print("Too big. Try again:")
    userNum = int(input())
print("You guessed correctly.")
```

9 **Test your code.** As a programmer, you should be sure that your code works before considering it finished.

- When programming in python, make sure that you get the indentations correct. Your code should look like this:

```
import random
print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
userNum = int(userNum)
while userNum != rightNum:
    if userNum < rightNum:
        print("Too small. Try again:")
    if userNum > rightNum:
        print("Too big. Try again:")
    userNum = int(input())
print("You guessed correctly.")
```

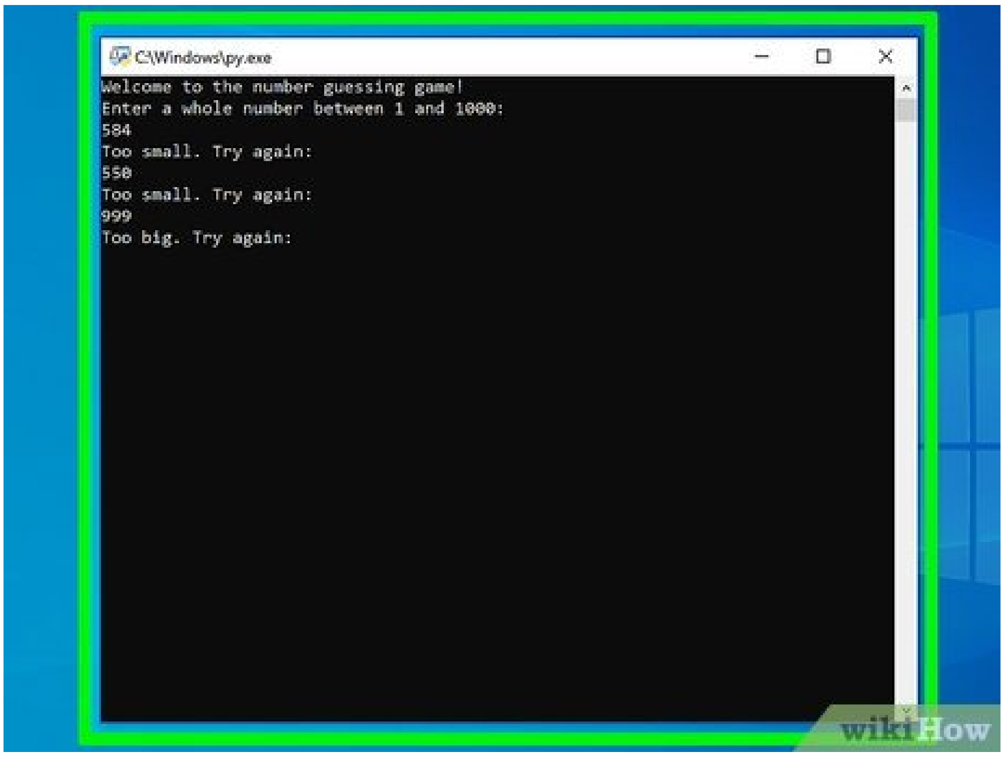


```
import random

def numInput():
    inp = input()
    while not inp.isdigit():
        print("You were told to enter a whole number! Enter a whole number:")
        inp = input()
    return int(inp)

print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
userNum = int(userNum)
while userNum != rightNum:
    if userNum < rightNum:
        print("Too small. Try again:")
    if userNum > rightNum:
        print("Too big. Try again:")
    userNum = int(input())
print("You guessed correctly.")
```

10 **Validate the input.** The player shouldn't be able to break your game by simply entering the wrong thing. "Validating the input" means making sure the player entered the correct thing before processing it.

- Open the game again and try entering anything that's not a number. The game will exit with a ValueError. To avoid this, you can implement a way to check whether the input was a number.
- Define a function. Since validating the input is quite long, and you have to do it multiple times, you should define a function. It will take no arguments and return a number. First, write `def numInput()`at the top of your code, directly under the import random.
- Get the player's input once. Use the `input()` function and assign the result to the variable `inp`.
- When the player's input is not a number, ask them to enter a number. To check whether a string is a number, use the `isdigit()`functions, which only allows a whole number, so you won't have to check for that separately.
- If the input is a number, convert it from string to number and return the result. Use the `int()` function for converting the string to an integer. This will make the conversion in the main code unnecessary, and you should remove it from there.
- Replace all calls to input() in the main code with calls to numInput().
- The code of the numInput() function will look like this:

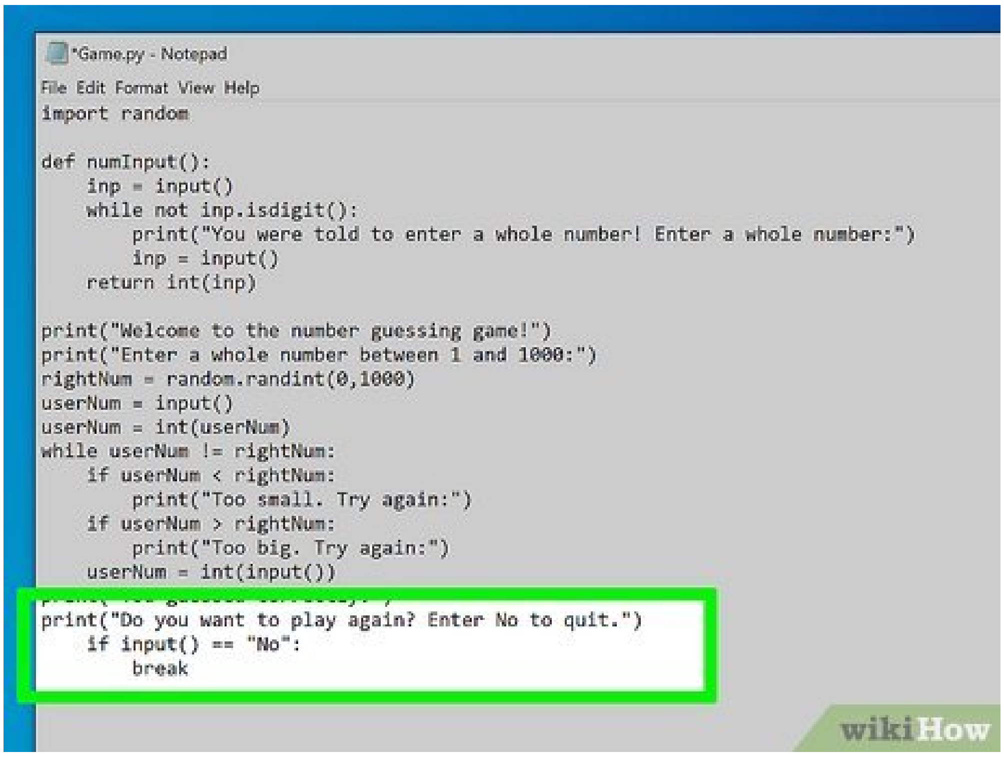
```
def numInput():
    inp = input()
    while not inp.isdigit():
        print("You were told to enter a whole number! Enter a whole number:")
        inp = input()
    return int(inp)
```


C:\Windows\py.exe

```
Welcome to the number guessing game!
Enter a whole number between 1 and 1000:
584
Too small. Try again:
550
Too small. Try again:
999
Too big. Try again:
```

11 **Test the game again.** Enter the wrong things on purpose to see what happens, and then fix any errors as they come up.

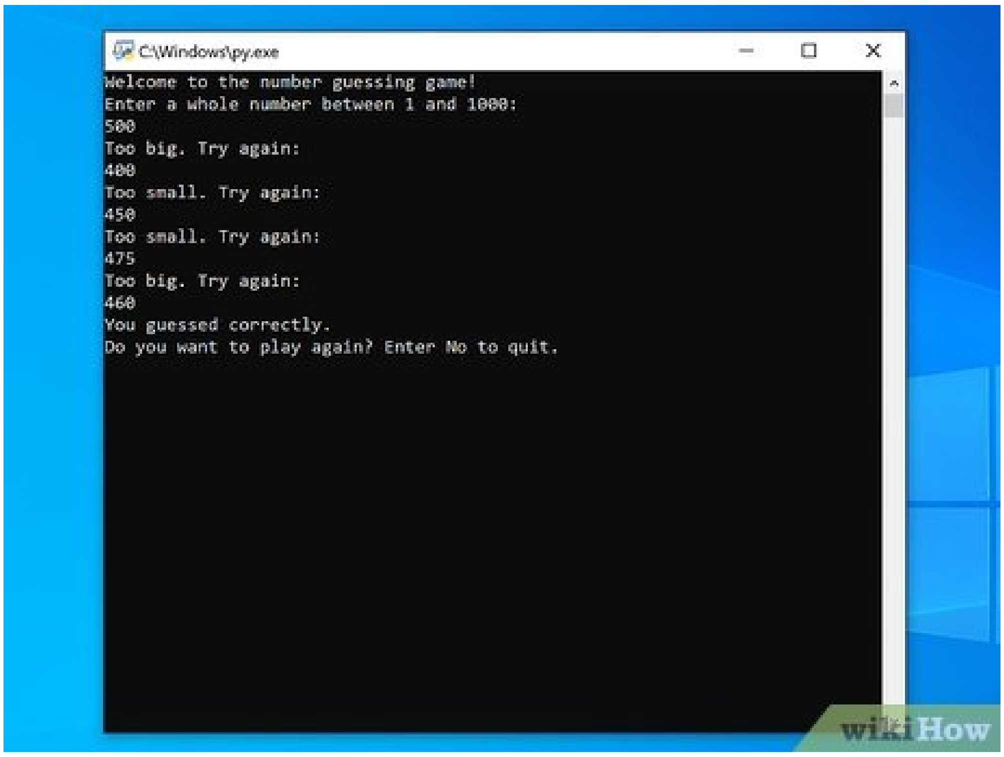
- Try entering some text when the program asks you for a number. Now, instead of exiting with an error message, the program will ask you for a number again.



```
import random

def numInput():
    inp = input()
    while not inp.isdigit():
        print("You were told to enter a whole number! Enter a whole number:")
        inp = input()
    return int(inp)

print("Welcome to the number guessing game!")
print("Enter a whole number between 1 and 1000:")
rightNum = random.randint(0,1000)
userNum = input()
userNum = int(userNum)
while userNum != rightNum:
    if userNum < rightNum:
        print("Too small. Try again:")
    if userNum > rightNum:
        print("Too big. Try again:")
    userNum = int(input())
print("Do you want to play again? Enter No to quit.")
    if input() == "No":
        break
```

12 **Suggest restarting the game when it finishes.** This way, the player could play your game for a longer time without having to constantly restart it.

- Put all code except the import and the function definition into a while-loop. Set `True` as the condition: this will always be true, so the loop will continue forever.
- Ask the player whether they want to play again after they guessed the number correctly. Use the `print()` function.
- If they answer "No", break out of the look. If they answer anything else, continue. Breaking out of a loop is done with the `break` statement.
- Move the "Welcome to the number guessing game" outside the while loop. The player probably doesn't want to be welcomed every time they play the game. Move the instruction print("Welcome to the number guessing game!" above the while True:, so it will be printed only once, when the user starts the first game.

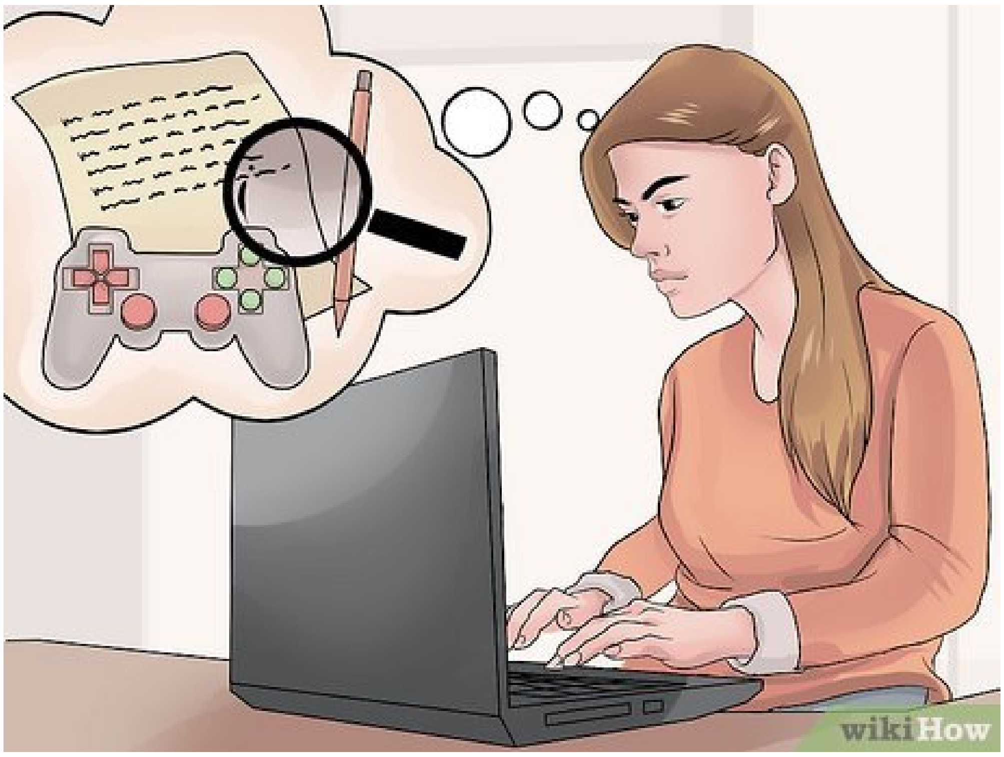


```
Welcome to the number guessing game!
Enter a whole number between 1 and 1000:
500
Too big. Try again:
400
Too small. Try again:
450
Too small. Try again:
475
Too big. Try again:
460
You guessed correctly.
Do you want to play again? Enter No to quit.
```

13 **Test the game.** You'll need to test your game each time you implement a new feature.

- Make sure to answer both "Yes" and "No" at least once to make sure that both options work. Here is what your code should look like:

```
import random

def numInput():
    inp = input()
    while not inp.isdigit():
        print("You were told to enter a whole number! Enter a whole number:")
        inp = input()
    return int(inp)

print("Welcome to the number guessing game!")
while True:
    print("Enter a whole number between 1 and 1000:")
    rightNum = random.randint(0,1000)
    userNum = numInput()
    while userNum != rightNum:
        if userNum < rightNum:
            print("Too small. Try again:")
        if userNum > rightNum:
            print("Too big. Try again:")
        userNum = numInput()
    print("You guessed correctly.")
    print("Do you want to play again? Enter No to quit.")
    if input() == "No":
        break
```



14 **Write other text-based games.** How about writing a text adventure next? Or a quiz game? Be creative.

Tip: It's sometimes helpful to look in the documentation if you're not sure how something is done or how a function is used. The Python 3 documentation is found at https://docs.python.org/3/. Sometimes searching for whatever you want to do on the internet also returns good results.

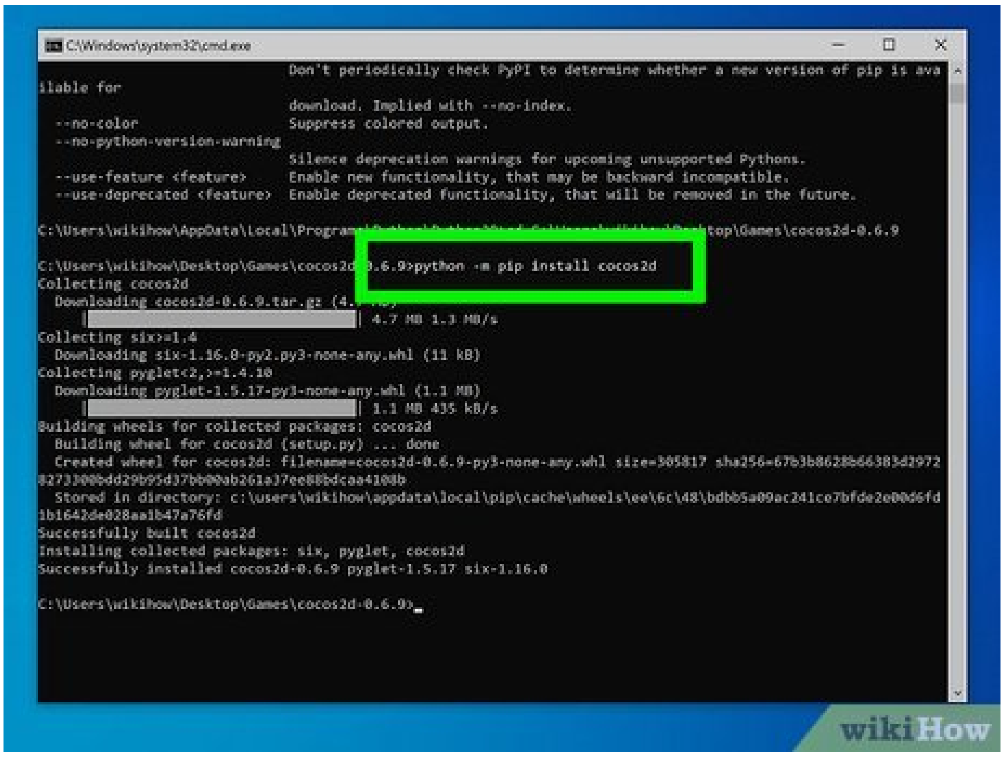
Part
2

Part 2 of 3: Making a Game with 2D Graphics

1 **Choose a graphics library.** Making graphics is very complicated, and most programming languages (including Python, C++, C, JavaScript) provide only minimal or even no support for graphics in the core or the standard libraries. So you'll have to use an external library to be able to make graphics, for example Pygame for Python.

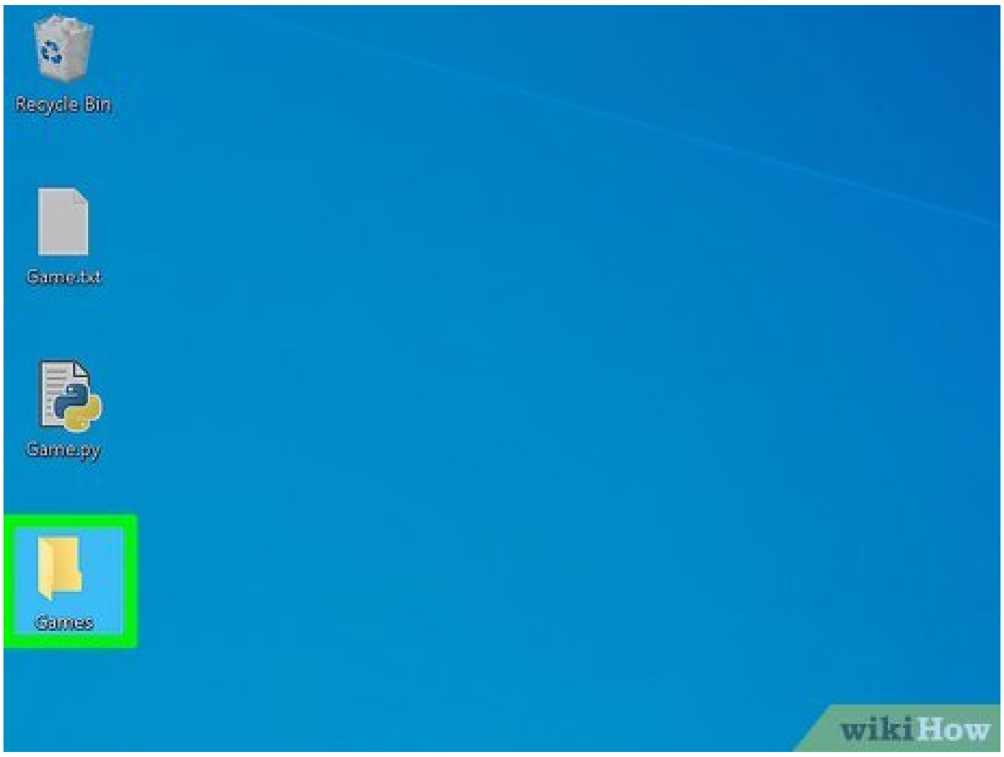
- Even with a graphics library, you'll have to worry about things like how to display a menu, how to check what the player clicked, how to display the tiles, and so on. If you'd rather focus on developing the actual game, you can use a game engine library like Unity, which implements these things easily.

This article will use Python with Cocos2D to show how to make a simple 2D platformer. Some of the mentioned concepts may not exist in other game engines. Refer to their documentation for more information.

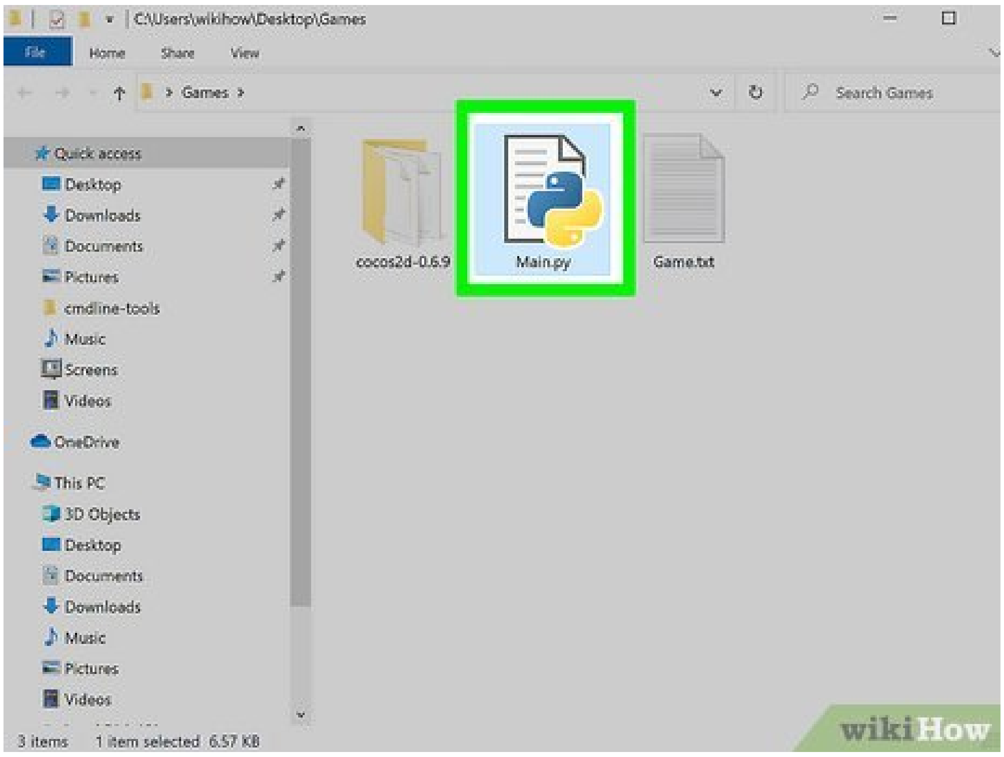


```
C:\Windows\system32\cmd.exe
                              Don't periodically check PyPI to determine whether a new version of pip is ava
ilable for
                              download. Implied with --no-index.
  --no-color                  Suppress colored output.
  --no-python-version-warning
                              Silence deprecation warnings for upcoming unsupported Pythons.
  --use-feature <feature>     Enable new functionality, that may be backward incompatible.
  --use-deprecated <feature>  Enable deprecated functionality, that will be removed in the future.

C:\Users\wikihow\AppData\Local\Program[illegible]top\Games\cocos2d-0.6.9

C:\Users\wikihow\Desktop\Games\cocos2d[illegible].6.9>python -m pip install cocos2d
Collecting cocos2d
  Downloading cocos2d-0.6.9.tar.gz (4.[illegible])
     |████████████████████████████████| 4.7 MB 1.3 MB/s
Collecting six>=1.4
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting pyglet<2,>=1.4.10
  Downloading pyglet-1.5.17-py3-none-any.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 435 kB/s
Building wheels for collected packages: cocos2d
  Building wheel for cocos2d (setup.py) ... done
  Created wheel for cocos2d: filename=cocos2d-0.6.9-py3-none-any.whl size=305817 sha256=67b3b8628b66383d2972
8273300bdd29b95d37bb00ab261a37ee88bdcaa4108b
  Stored in directory: c:\users\wikihow\appdata\local\pip\cache\wheels\ee\6c\48\bdbb5a09ac241ce7bfde2e00d6fd
1b1642de028aa1b47a76fd
Successfully built cocos2d
Installing collected packages: six, pyglet, cocos2d
Successfully installed cocos2d-0.6.9 pyglet-1.5.17 six-1.16.0

C:\Users\wikihow\Desktop\Games\cocos2d-0.6.9>_
```

2 **Install the graphics library you chose.** Cocos2D for Python is easy to install. You can get it from http://python.cocos2d.org/index.html, or by running `sudo pip3 install cocos2d` if you're using Linux.

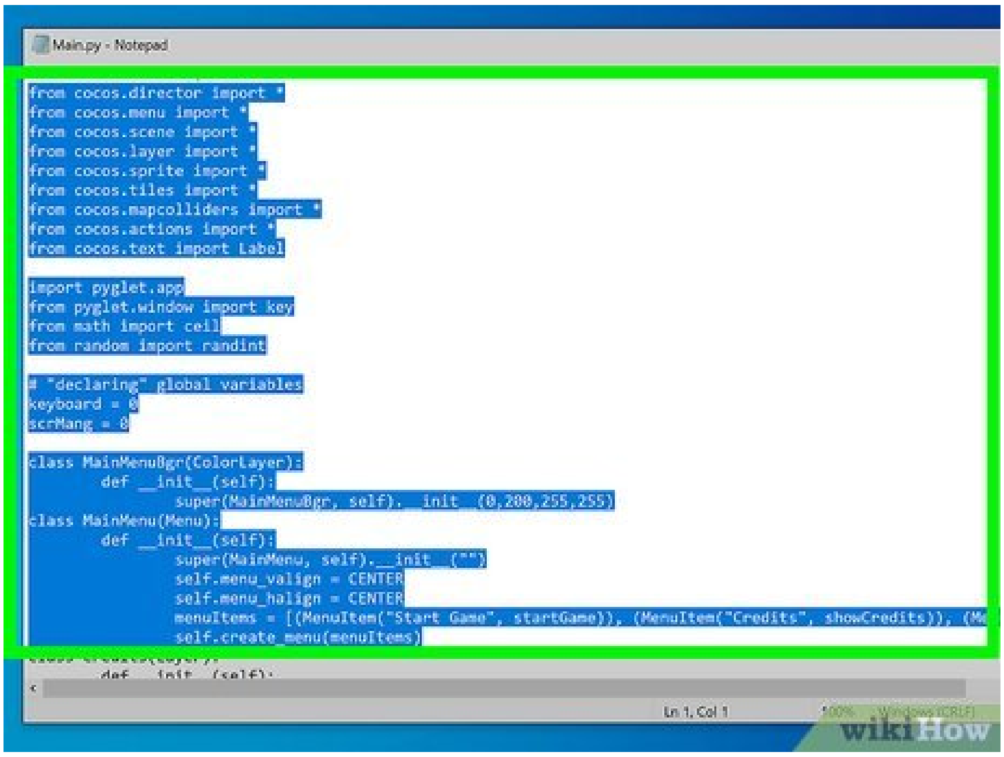


3 **Make a new directory for your game and media.** You will use things like images and sounds in your game. Keep these things in the same directory as the program. This directory shouldn't contain anything else so that you can easily see what assets you have in the game.



4 **Create a new code file in the new directory.** Call it main, with the file extension for your programming language. If you write a large and complex program where it makes sense to have multiple program files, this will show you which is the main file.

- In this example, we'll create a file called `main.py` that will contain all of our code.



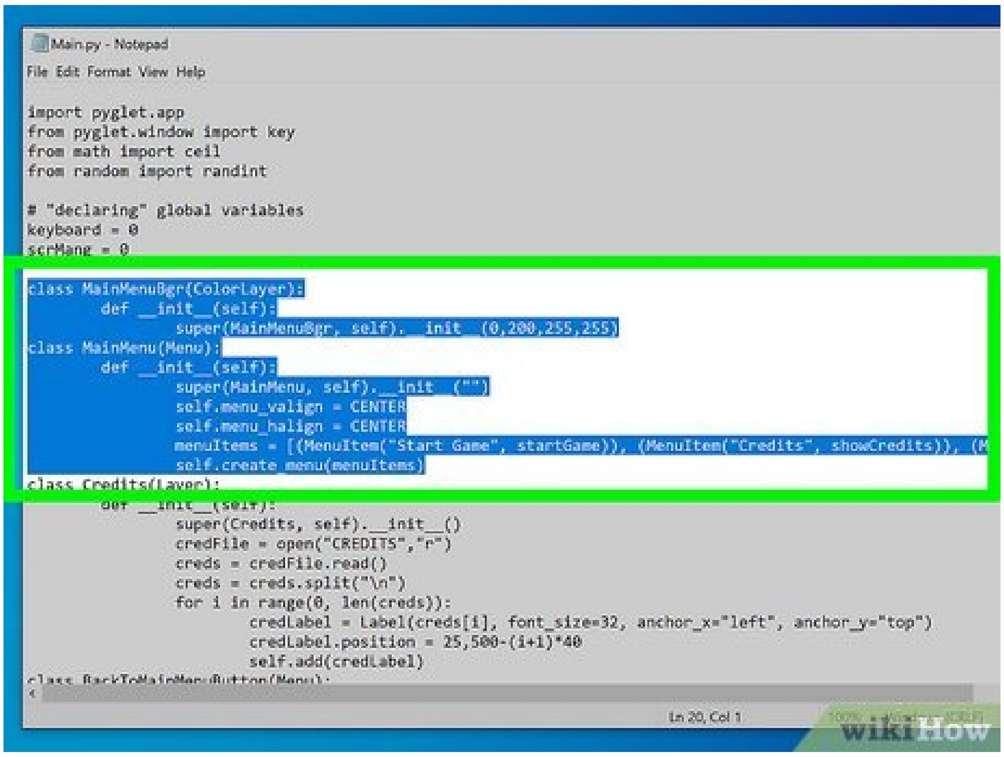
```
from cocos.director import *
from cocos.menu import *
from cocos.scene import *
from cocos.layer import *
from cocos.sprite import *
from cocos.tiles import *
from cocos.mapcolliders import *
from cocos.actions import *
from cocos.text import Label

import pyglet.app
from pyglet.window import key
from math import ceil
from random import randint

# "declaring" global variables
keyboard = 0
scrMang = 0

class MainMenuBgr(ColorLayer):
        def __init__(self):
                super(MainMenuBgr, self).__init__(0,200,255,255)
class MainMenu(Menu):
        def __init__(self):
                super(MainMenu, self).__init__("")
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Start Game", startGame)), (MenuItem("Credits", showCredits)), (Me
                self.create_menu(menuItems)
```

5 **Create the game window.** This is the basic prerequisite for a game with graphics.

- Import the necessary cocos2d sub-modules: cocos.director, cocos.scene and cocos.layer. This is done with `from subModuleName import *`, where sub-Module name is the submodule you want to import. The difference between from ... import * and import ... is that you don't have to put the module name in front of everything you use from that module with the former.
- Define a subclass `MainMenuBgr` of the ColorLayer. This basically means that any main menu background you create will behave like a color layer with some changes you make.

- Start the cocos director. This will give you a new window. If you don't set a caption, the window will have the same caption as the file name (main.py), which won't look professional. Allow the window to be resized with by setting resizable to True.
- Define a function showMainMenu. You should put the code for showing the main menu into a function because this will allow you to easily return to the main menu by calling the function again.
- Create a scene. The scene consists of one layer for now, which is an object of the MainMenuBgr class you defined.
- Run this scene in the window.

```
from cocos.director import *
from cocos.scene import *
from cocos.layer import *

class MainMenuBgr(ColorLayer):
        def __init__(self):
                super(MainMenu, self).__init__(0,200,255,255)

def showMainMenu():
        menuSc = Scene(MainMenuBgr())
        director.run(menuSc)

director.init(caption="IcyPlat - a simple platformer", resizable=True)
showMainMenu()
```


Main.py - Notepad

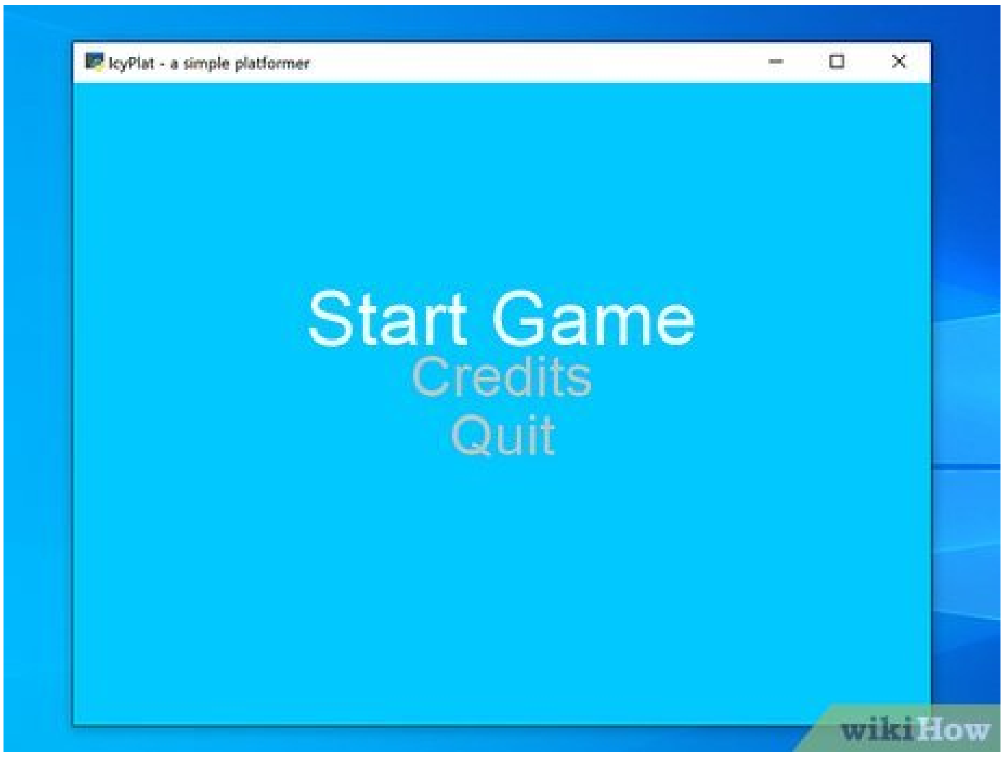
```
import pyglet.app
from pyglet.window import key
from math import ceil
from random import randint

# "declaring" global variables
keyboard = 0
scrMang = 0

class MainMenuBgr(ColorLayer):
        def __init__(self):
                super(MainMenuBgr, self).__init__(0,200,255,255)
class MainMenu(Menu):
        def __init__(self):
                super(MainMenu, self).__init__("")
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Start Game", startGame)), (MenuItem("Credits", showCredits)), (M
                self.create_menu(menuItems)
class Credits(Layer):
        def __init__(self):
                super(Credits, self).__init__()
                credFile = open("CREDITS","r")
                creds = credFile.read()
                creds = creds.split("\n")
                for i in range(0, len(creds)):
                        credLabel = Label(creds[i], font_size=32, anchor_x="left", anchor_y="top")
                        credLabel.position = 25,500-(i+1)*40
                        self.add(credLabel)
class BackToMainMenuButton(Menu):
```

6 **Add a main menu to the window.** Besides the actual game, you'll need to add a menu the player can use to close the window, among other elements you can add later.

- Import cocos.menu (again with the from instruction) and pyglet.app (this time with import).
- Define MainMenu as a subclass of Menu.
- Set the alignment of the main menu. You have to set the vertical and horizontal alignment separately.

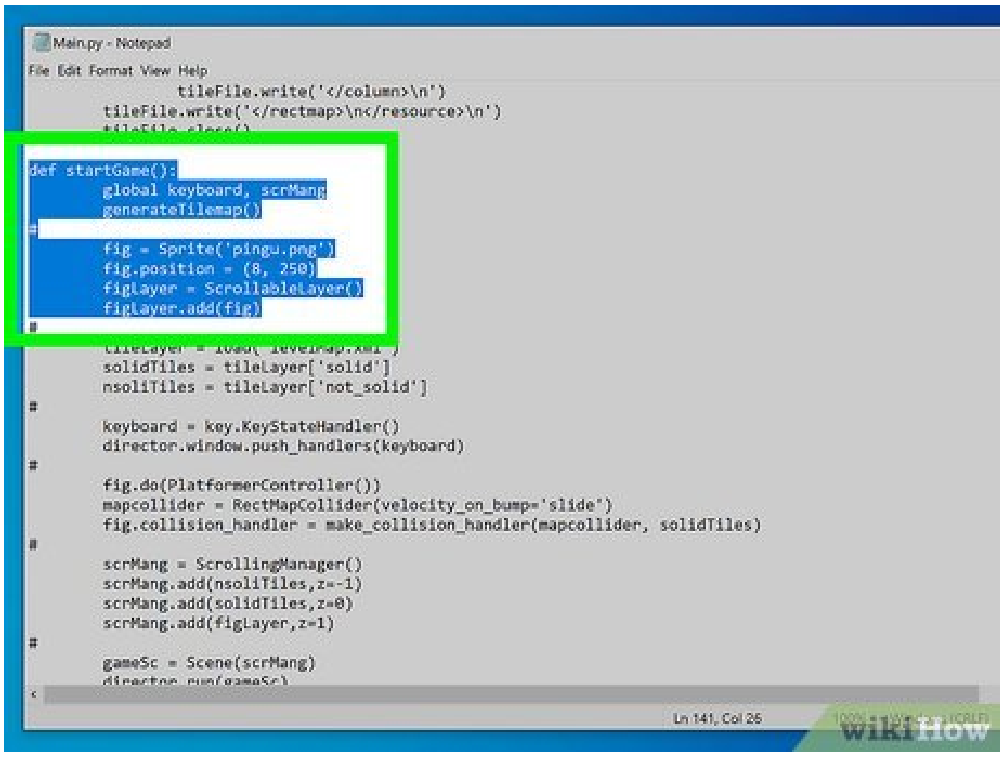
- Create a list of menu items and add them to the menu. You should have the menu items "Start Game" and "Quit" at the very least. Every menu item should be placed inside of brackets. Each item must have a label and a callback function that determines what happens when the player clicks it. For the "Start Game" item, use the `startGame` function (you'll write it soon), for the "Quit" item, use "pyglet.app.exit" (already exists). Create the actual menu by calling `self.create_menu(menuItems)`
- Define `startGame()` Just put `pass` into the definition for now, you'll replace that when you write the actual game.
- Go to the place in your code where you created the menuSc scene, and add a MainMenu object to it.
- Your entire code should now look as follows:

```
from cocos.director import *
from cocos.menu import *
from cocos.scene import *
from cocos.layer import *

import pyglet.app

class MainMenuBgr(ColorLayer):
        def __init__(self):
                super(MainMenuBgr, self).__init__(0,200,255,255)
class MainMenu(Menu):
        def __init__(self):
                super(MainMenu, self).__init__("")
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Start Game", startGame)), (MenuItem("Quit", pygl
et.app.exit))]
                self.create_menu(menuItems)

def startGame():
        pass

def showMainMenu():
        menuSc = Scene(MainMenuBgr())
        menuSc.add(MainMenu())
        director.run(menuSc)
director.init(caption="IcyPlat - a simple platformer", resizable=True)
showMainMenu()
```



7 **Test your code.** Test the code early, while it's still short and relatively simple. Then you can identify and correct any mistakes in the basic structure before things get too complicated.

- The code from the instructions should open a window captioned "IcyPlat - a simple platformer." The background is light blue and you can resize the window. When you click "Start Game" on the menu, nothing should happen (yet). When you click "Quit," the window will close.



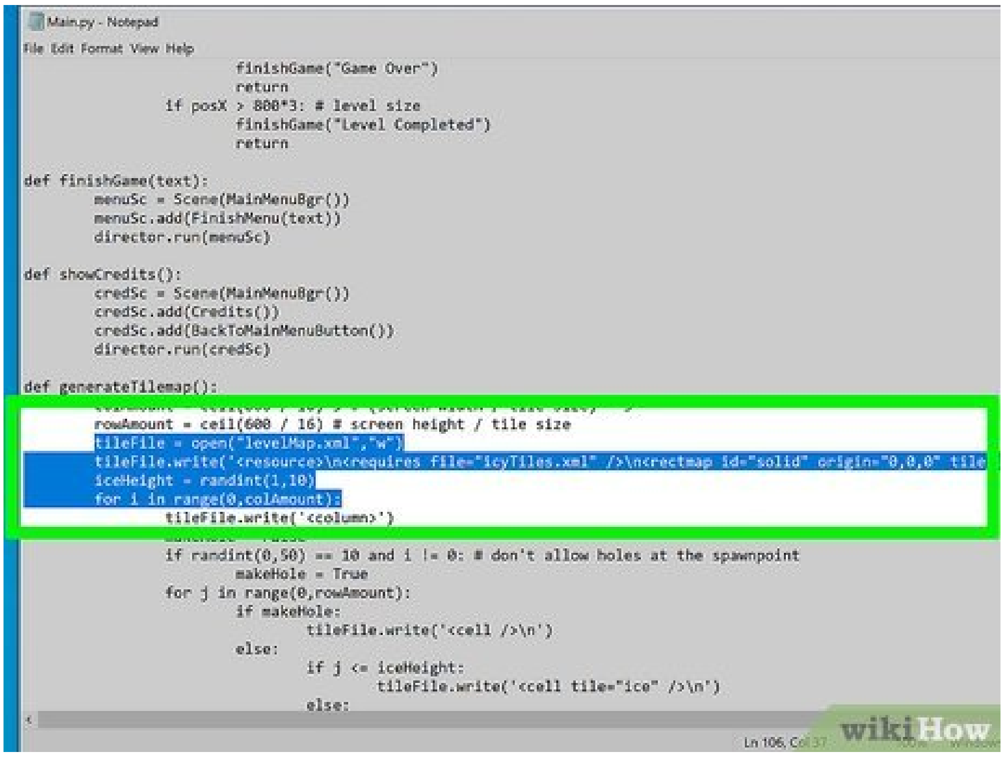
```
            tileFile.write('</column>\n')
        tileFile.write('</rectmap>\n</resource>\n')
        tileFile.close()

def startGame():
        global keyboard, scrMang
        generateTilemap()
#
        fig = Sprite('pingu.png')
        fig.position = (8, 250)
        figLayer = ScrollableLayer()
        figLayer.add(fig)
#
        tileLayer = load('levelmap.xml')
        solidTiles = tileLayer['solid']
        nsoliTiles = tileLayer['not_solid']
#
        keyboard = key.KeyStateHandler()
        director.window.push_handlers(keyboard)
#
        fig.do(PlatformerController())
        mapcollider = RectMapCollider(velocity_on_bump='slide')
        fig.collision_handler = make_collision_handler(mapcollider, solidTiles)
#
        scrMang = ScrollingManager()
        scrMang.add(nsoliTiles,z=-1)
        scrMang.add(solidTiles,z=0)
        scrMang.add(figLayer,z=1)
#
        gameSc = Scene(scrMang)
        director.run(gameSc)
```

8 **Create a sprite.** A sprite is a "game object," or a 2-dimensional image. Sprites can be in-game objects, icons, background decorations, characters, and anything else you can represent with an image in the game. We'll start by creating a sprite for a character that the player can interact with.

- Import the cocos.sprite submodule with the from-import-expression.
- Find an image to represent the sprite. You can't display a sprite if you don't have a picture for it. You can draw one, or you can get one from the internet (watch out for the licenses, though, if you're planning to publish your game). For this example, head to https://opengameart.org/content/tux-classic-hero-style and save the PNG image of running penguins to your computer. Then, crop out one of the running penguins, since you'll only need one for now.
- Create a layer as a new object of the ScrollableLayer class. Then, create the sprite as a Sprite object and set its position to (8, 250). For reference, the point (0, 0) is in the bottom left corner. This is quite high, but it will make sure that the penguin doesn't get stuck in the ice.
- Add the sprite to the sprite's layer.
- Create a new scene out of the sprite's layer and run it.

```
def startGame():
        figLayer = ScrollableLayer()
        fig = Sprite('pingu.png')
        fig.position = (75, 100)
        figLayer.add(fig)
#
        gameSc = Scene(figLayer)
        director.run(gameSc)
```

- Run the code. You should see a small penguin figure (or whatever you drew) on a black background after you click **Start Game**.



```
                        finishGame("Game Over")
                        return
                if posX > 800*3: # level size
                        finishGame("Level Completed")
                        return

def finishGame(text):
        menuSc = Scene(MainMenuBgr())
        menuSc.add(FinishMenu(text))
        director.run(menuSc)

def showCredits():
        credSc = Scene(MainMenuBgr())
        credSc.add(Credits())
        credSc.add(BackToMainMenuButton())
        director.run(credSc)

def generateTilemap():
        rowAmount = ceil(600 / 16) # screen height / tile size
        tileFile = open("levelMap.xml","w")
        tileFile.write('<resource>\n<requires file="icyTiles.xml" />\n<rectmap id="solid" origin="0,0,0" tile
        iceHeight = randint(1,10)
        for i in range(0,colAmount):
                tileFile.write('<column>')
                if randint(0,50) == 10 and i != 0: # don't allow holes at the spawnpoint
                        makeHole = True
                for j in range(0,rowAmount):
                        if makeHole:
                                tileFile.write('<cell />\n')
                        else:
                                if j <= iceHeight:
                                        tileFile.write('<cell tile="ice" />\n')
                                else:
```

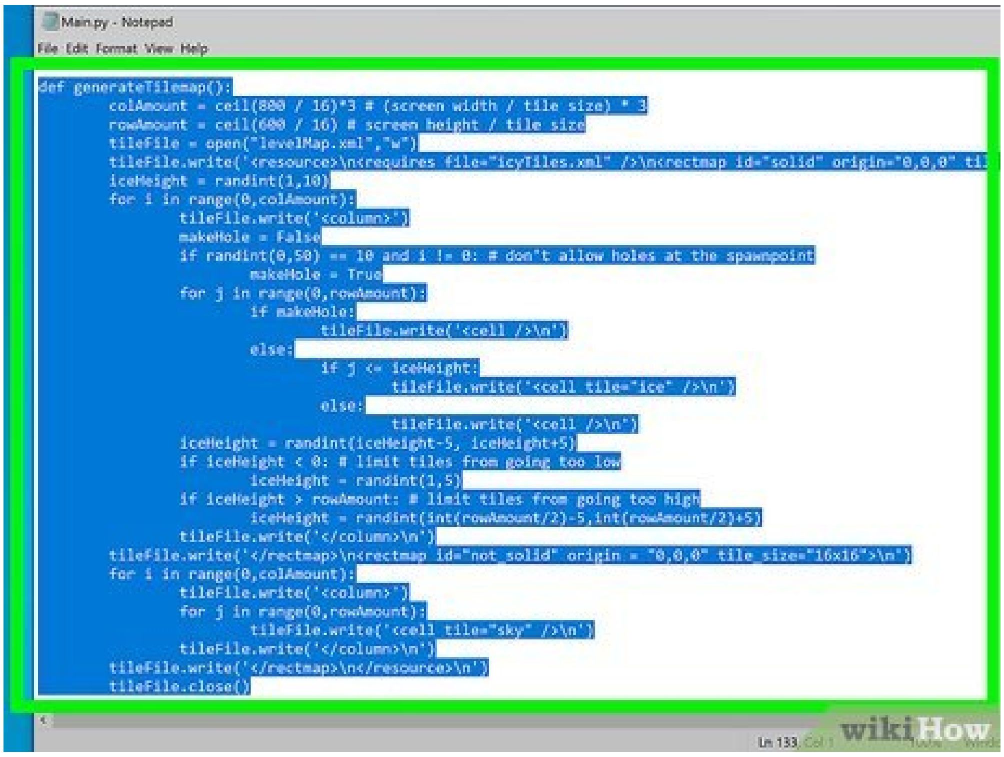
9 **Dream up your landscape.** In most games, your sprites shouldn't just float in the void. They should actually stand on some surface, with something around them. In 2D games, this is often done with a tile set and a tile map. The tile set basically says what kind of surface squares and background squares exist, and what they look like.

- Create a tile set. The tile set for this game will be very basic: one tile for ice and one tile for sky. The ice tile used in this example is from here, under CC-BY-SA 3.0.

- Create a tile set picture. That's a picture of all tiles, which have to all be of the same size (edit them if they aren't) and have the size you want to see in the game, next to each other. Save your picture as `icyTiles.png`

- Create the tile set description. That's an XML file. The XML file contains information on how big the tiles are in the tile set picture, which picture to use, and where to find which tile there. Create an XML file named `icyTiles.xml` with the code below:

```
<?xml version="1.0"?>
<resource>
    <imageatlas size="16x16" file="icyTiles.png">
        <image id="i-ice" offset="0,0" />
        <image id="i-sky" offset="16,0" />
    </imageatlas>
    <tileset>
        <tile id="ice"><image ref="i-ice" />
        </tile>
        <tile id="sky"><image ref="i-sky" />
        </tile>
    </tileset>
</resource>
```

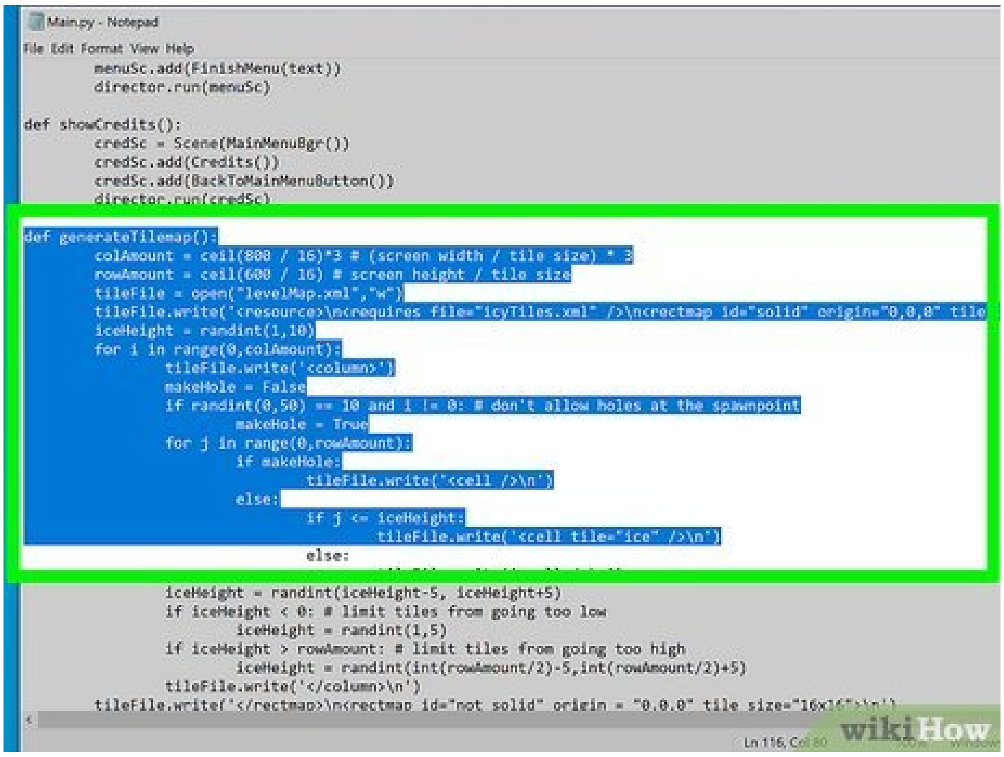


```
def generateTilemap():
        colAmount = ceil(800 / 16)*3 # (screen width / tile size) * 3
        rowAmount = ceil(600 / 16) # screen height / tile size
        tileFile = open("levelMap.xml","w")
        tileFile.write('<resource>\n<requires file="icyTiles.xml" />\n<rectmap id="solid" origin="0,0,0" til
        iceHeight = randint(1,10)
        for i in range(0,colAmount):
                tileFile.write('<column>')
                makeHole = False
                if randint(0,50) == 10 and i != 0: # don't allow holes at the spawnpoint
                        makeHole = True
                for j in range(0,rowAmount):
                        if makeHole:
                                tileFile.write('<cell />\n')
                        else:
                                if j <= iceHeight:
                                        tileFile.write('<cell tile="ice" />\n')
                                else:
                                        tileFile.write('<cell />\n')
                iceHeight = randint(iceHeight-5, iceHeight+5)
                if iceHeight < 0: # limit tiles from going too low
                        iceHeight = randint(1,5)
                if iceHeight > rowAmount: # limit tiles from going too high
                        iceHeight = randint(int(rowAmount/2)-5,int(rowAmount/2)+5)
                tileFile.write('</column>\n')
        tileFile.write('</rectmap>\n<rectmap id="not_solid" origin = "0,0,0" tile_size="16x16">\n')
        for i in range(0,colAmount):
                tileFile.write('<column>')
                for j in range(0,rowAmount):
                        tileFile.write('<cell tile="sky" />\n')
                tileFile.write('</column>\n')
        tileFile.write('</rectmap>\n</resource>\n')
        tileFile.close()
```

10 **Make a tile map for your landscape.** A tile map is a map that defines which tile is at which position in your level. In the example, you should define a function to generate tile maps because designing tile maps by hand is very tedious. A more advanced game would usually have some sort of level editor, but for becoming familiar with 2D game development, an algorithm can provide good enough levels.

- Find out how many rows and columns are needed. For this, divide the screen size by the tile size both horizontally (columns) and vertically (rows). Round the number upwards; you need a function of the math module for that, so add `from math import ceil` to the imports at the top of your code.
- Open a file for writing. This will erase all previous content of the file, so choose a name that no file in the directory has yet, like `levelMap.xml`
- Write the opening tags into the file.
- Generate a tile map according to the algorithm. You use the one in the code below, or you can come up with one on your own. Make sure to import the randint function from the module random: it's required for the code below to work, and whatever you come up with will probably also need random integers. Also, make sure to put sky tiles and ice tiles in different layers: ice is solid, sky is not.
- Write the closing tags into the file and close the file.

```
def generateTilemap():
        colAmount = ceil(800 / 16)*3 # (screen width / tile size) * 3
        rowAmount = ceil(600 / 16) # screen height / tile size
        tileFile = open("levelMap.xml","w")
        tileFile.write('<resource>\n<requires file="icyTiles.xml" />\n<rectmap id="solid" ori
gin="0,0,1" tile_size="16x16">\n')
        iceHeight = randint(1,10)
        for i in range(0,colAmount):
                tileFile.write('<column>')
                makeHole = False
                if randint(0,50) == 10 and i != 0: # don't allow holes at the spawnpoint
                        makeHole = True
                for j in range(0,rowAmount):
                        if makeHole:
                                tileFile.write('<cell />\n')
                        else:
                                if j <= iceHeight:
                                        tileFile.write('<cell tile="ice" />\n')
                                else:
                                        tileFile.write('<cell />\n')
                iceHeight = randint(iceHeight-5, iceHeight+5)
                if iceHeight < 0: # limit tiles from going too low
                        iceHeight = randint(1,5)
                if iceHeight > rowAmount: # limit tiles from going too high
                        iceHeight = randint(int(rowAmount/2)-5,int(rowAmount/2)+5)
                tileFile.write('</column>\n')
        tileFile.write('</rectmap>\n<rectmap id="not_solid" origin = "0,0,0" tile_size="16x1
6">\n')
        for i in range(0,colAmount):
                tileFile.write('<column>')
                for j in range(0,rowAmount):
                        tileFile.write('<cell tile="sky" />\n')
                tileFile.write('</column>\n')
        tileFile.write('</rectmap>\n</resource>\n')
        tileFile.close()
```

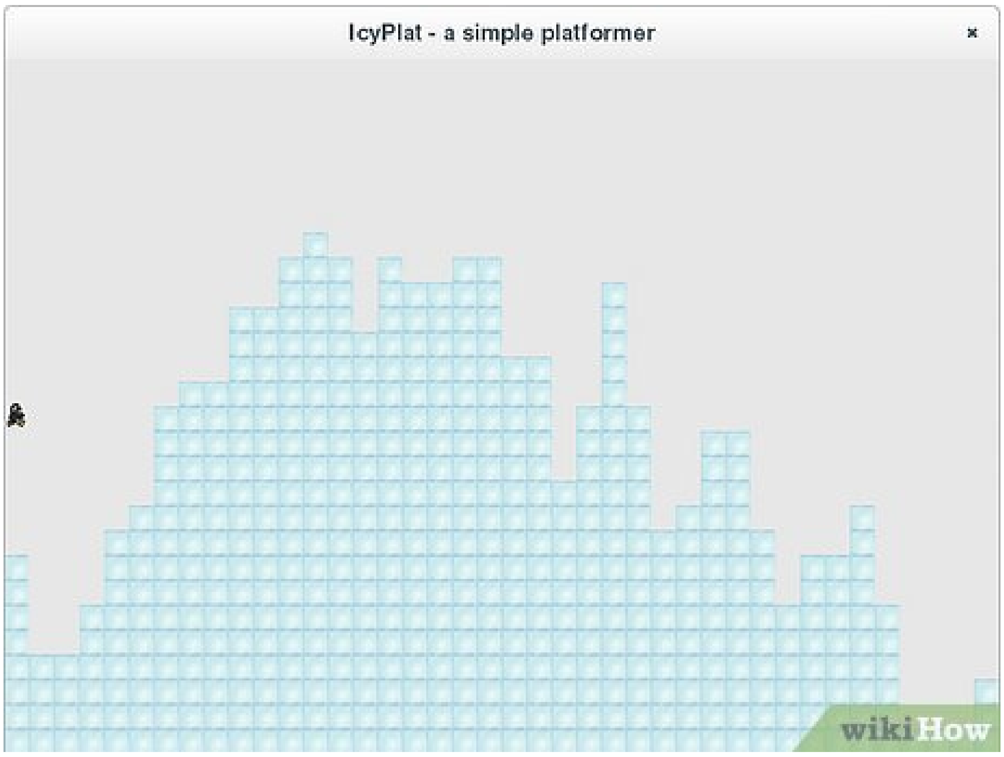


```
Main.py - Notepad
File  Edit  Format  View  Help
        menuSc.add(FinishMenu(text))
        director.run(menuSc)

def showCredits():
        credSc = Scene(MainMenuBgr())
        credSc.add(Credits())
        credSc.add(BackToMainMenuButton())
        director.run(credSc)

def generateTilemap():
        colAmount = ceil(800 / 16)*3 # (screen width / tile size) * 3
        rowAmount = ceil(600 / 16) # screen height / tile size
        tileFile = open("levelMap.xml","w")
        tileFile.write('<resource>\n<requires file="icyTiles.xml" />\n<rectmap id="solid" origin="0,0,0" tile
        iceHeight = randint(1,10)
        for i in range(0,colAmount):
                tileFile.write('<column>')
                makeHole = False
                if randint(0,50) == 10 and i != 0: # don't allow holes at the spawnpoint
                        makeHole = True
                for j in range(0,rowAmount):
                        if makeHole:
                                tileFile.write('<cell />\n')
                        else:
                                if j <= iceHeight:
                                        tileFile.write('<cell tile="ice" />\n')
                                else:
                iceHeight = randint(iceHeight-5, iceHeight+5)
                if iceHeight < 0: # limit tiles from going too low
                        iceHeight = randint(1,5)
                if iceHeight > rowAmount: # limit tiles from going too high
                        iceHeight = randint(int(rowAmount/2)-5,int(rowAmount/2)+5)
                tileFile.write('</column>\n')
        tileFile.write('</rectmap>\n<rectmap id="not solid" origin = "0.0.0" tile size="16x16">\n')
Ln 116, Col 80
```

11 **Display the tile map.** Import everything from `cocos.tiles` and then go into the startGame function for that.

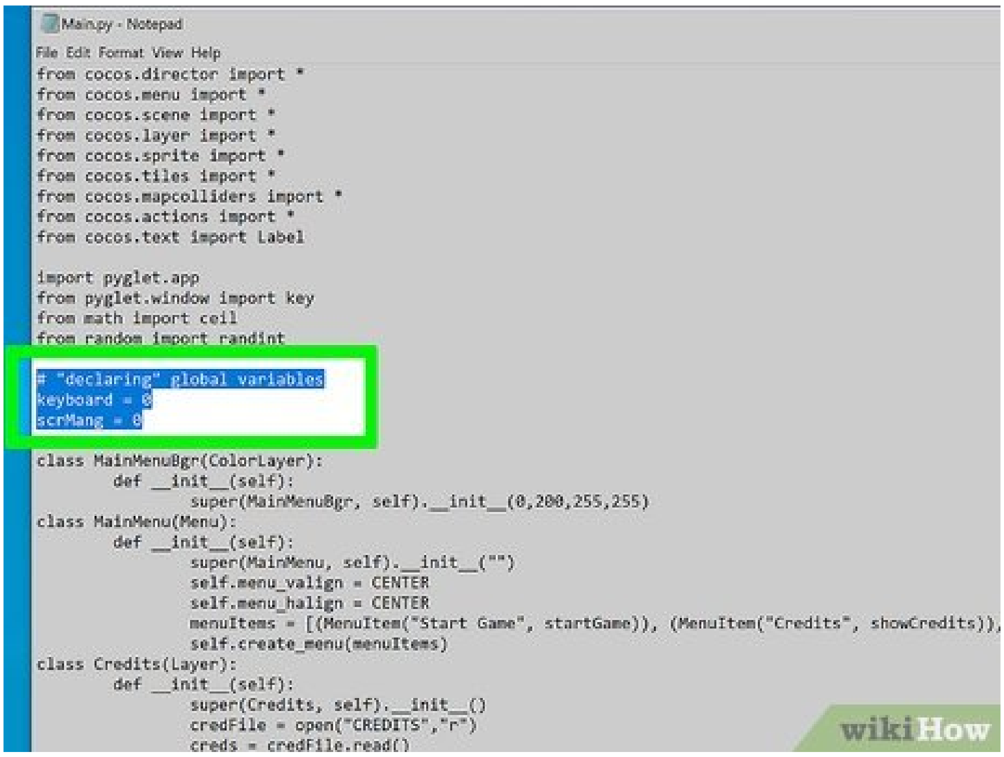
- At the beginning of your startGame function, generate a tile map using the function you defined for that.
- Create a new scrolling manager. Do this directly under the line where you add the sprite to its layer.
- Create a new layer containing the tiles, which will be loaded from the levelMap.xml tile map your generateTilemap function generated.
- Add the non-solid layer, the solid layer and the sprite layer to the scrolling manager, exactly in this order. You can add a z-position if you want.
- Instead of creating the scene from the sprite layer, create it from the scrolling manager.
- Your startGame function should now look like this:

```
def startGame():
        generateTilemap()
#
        fig = Sprite('pingu.png')
        fig.position = (8, 500)
        figLayer = ScrollableLayer()
        figLayer.add(fig)
#
        tileLayer = load('levelMap.xml')
        solidTiles = tileLayer['solid']
        nsoliTiles = tileLayer['not_solid']
#
        scrMang = ScrollingManager()
        scrMang.add(nsoliTiles,z=-1)
        scrMang.add(solidTiles,z=0)
        scrMang.add(figLayer,z=1)
#
        gameSc = Scene(scrMang)
        director.run(gameSc)
```



12 **Test your code.** You should test your code often to make sure that the new features you implemented really work.

- The code in the example should now show some icy landscape behind the penguin. If the penguin looks like it is hovering far over the ice, you didn't do anything wrong, and it will be fixed in the next step.



```
from cocos.director import *
from cocos.menu import *
from cocos.scene import *
from cocos.layer import *
from cocos.sprite import *
from cocos.tiles import *
from cocos.mapcolliders import *
from cocos.actions import *
from cocos.text import Label

import pyglet.app
from pyglet.window import key
from math import ceil
from random import randint

# "declaring" global variables
keyboard = 0
scrMang = 0

class MainMenuBgr(ColorLayer):
        def __init__(self):
                super(MainMenuBgr, self).__init__(0,200,255,255)
class MainMenu(Menu):
        def __init__(self):
                super(MainMenu, self).__init__("")
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Start Game", startGame)), (MenuItem("Credits", showCredits)),
                self.create_menu(menuItems)
class Credits(Layer):
        def __init__(self):
                super(Credits, self).__init__()
                credFile = open("CREDITS","r")
                creds = credFile.read()
```

13 **Add the controls.** The player has many more ways to interact with the program in a 2D game than in a text-based game. A common one includes moving their figure when the correct key is pressed.

- Import everything from `cocos.mapcolliders` and from `cocos.actions`. Also import `key` from `pyglet.window`.

- "Declare" some global variables. Global variables are shared between functions. You can't really declare variables in Python, but you have to say that a global variable exists in the main code before using it. You can assign 0 as the value because a function will take care of assigning the correct value later. So add under the import expressions:

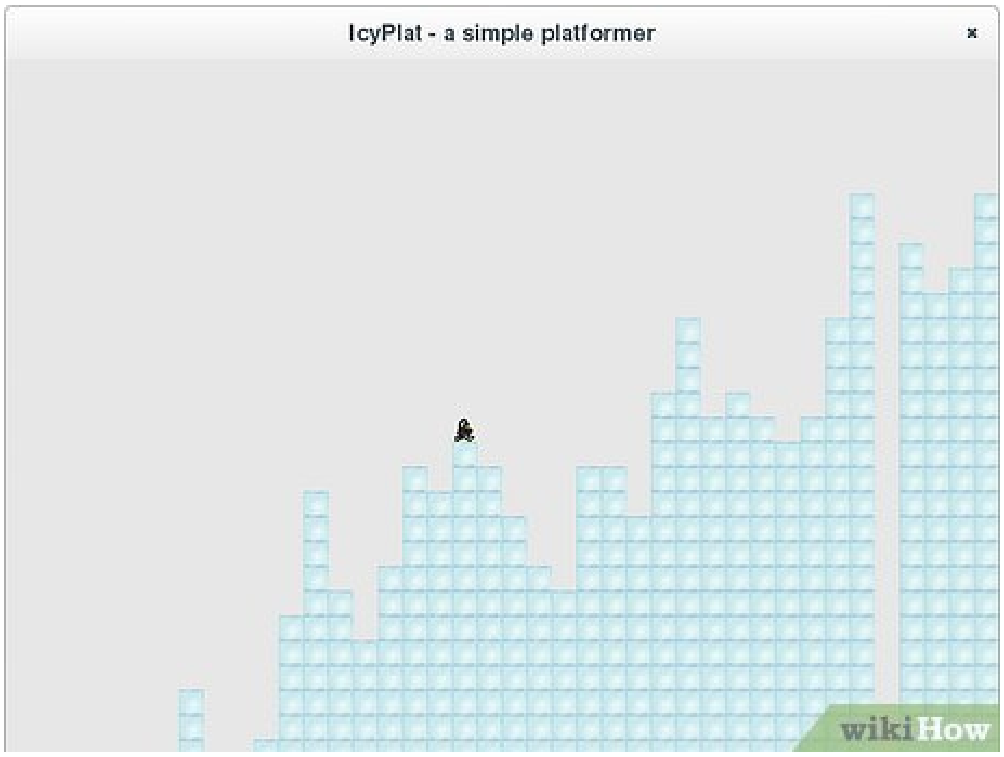
```
# "declaring" global variables
keyboard = 0
scrMang = 0
```

- Adjust your `startGame` function:
  - Say that you use the global variables `keyboard` and `scrMang`. Do this by writing `global keyboard, scrMang` at the top of the function.
  - Make the window listen to keyboard events.
  - Tell the figure to act based on a `PlatformerController`. You'll implement that `PlatformerController` soon.
  - Create a map collider to handle collisions between the solid tiles and the figure.

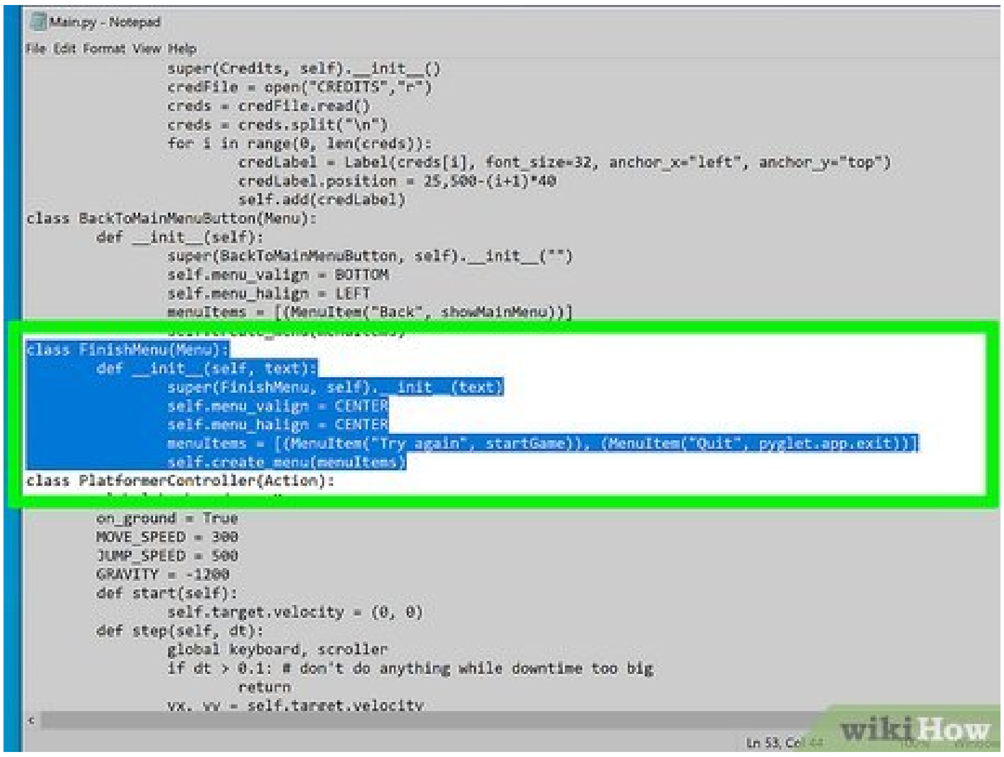
```
def startGame():
        global keyboard, scrMang
        generateTilemap()
#
        fig = Sprite('pingu.png')
        fig.position = (8, 250)
        figLayer = ScrollableLayer()
        figLayer.add(fig)
#
        tileLayer = load('levelMap.xml')
        solidTiles = tileLayer['solid']
        nsoliTiles = tileLayer['not_solid']
#
        keyboard = key.KeyStateHandler()
        director.window.push_handlers(keyboard)
#
        fig.do(PlatformerController())
        mapcollider = RectMapCollider(velocity_on_bump='slide')
        fig.collision_handler = make_collision_handler(mapcollider, solidTiles)
#
        scrMang = ScrollingManager()
        scrMang.add(nsoliTiles,z=-1)
        scrMang.add(solidTiles,z=0)
        scrMang.add(figLayer,z=1)
#
        gameSc = Scene(scrMang)
        director.run(gameSc)
```

- Create a platformer controller. This is what will move the figure according to your keypresses.
    - Define the platformer controller as a subclass of `Action`.
    - Define the move speed, the jump speed and the gravity.
    - Define the `start` function. This function is called once, when the platformer controller is connected to the figure. It should set its speed to 0 both in x and in y direction.
    - Define the `step` function. It will be repeated while the scene is running.
    - Tell the `step` function to use the global variables `keyboard` and `scrMang`.
    - Get and change the velocity. Save the x and the y velocity in separate variables. Set the x velocity to either 1 or -1 (depending on whether the left or right key was pressed) multiplied with the move speed. Add gravity to the y velocity. Multiply it with downtime so it works the same way on slower devices. If the space key is pressed and the figure is standing on the ground, jump by changing y velocity to jump speed.
    - Calculate to where the figure should move. Then let the collision handler adjust that position if it is inside of a solid tile. Finally, move the figure to the new adjusted position.
    - Set the focus of the scrolling manager on the figure. This causes the camera to move in a reasonable way when the figure moves.

```
class PlatformerController(Action):
    global keyboard, scrMang
    on_ground = True
    MOVE_SPEED = 300
    JUMP_SPEED = 500
    GRAVITY = -1200
    def start(self):
        self.target.velocity = (0, 0)
    def step(self, dt):
        global keyboard, scroller
        if dt > 0.1: # don't do anything while downtime to big
            return
        vx, vy = self.target.velocity
        vx = (keyboard[key.RIGHT] - keyboard[key.LEFT]) * self.MOVE_SPEED
        vy += self.GRAVITY * dt
        if self.on_ground and keyboard[key.SPACE]:
            vy = self.JUMP_SPEED
        dx = vx * dt
        dy = vy * dt
        last = self.target.get_rect()
        new = last.copy()
        new.x += dx
        new.y += dy
        self.target.velocity = self.target.collision_handler(last, new, vx, vy)
        self.on_ground = (new.y == last.y)
        self.target.position = new.center
        scrMang.set_focus(*new.center)
```



14 **Test your code.** If you followed the example, you should now be able to move the penguin with the arrow keys and jump by pressing the space bar. Also, the penguin should now fall down instead of hovering over the ground.



```
                super(Credits, self).__init__()
                credFile = open("CREDITS","r")
                creds = credFile.read()
                creds = creds.split("\n")
                for i in range(0, len(creds)):
                        credLabel = Label(creds[i], font_size=32, anchor_x="left", anchor_y="top")
                        credLabel.position = 25,500-(i+1)*40
                        self.add(credLabel)
class BackToMainMenuButton(Menu):
        def __init__(self):
                super(BackToMainMenuButton, self).__init__("")
                self.menu_valign = BOTTOM
                self.menu_halign = LEFT
                menuItems = [(MenuItem("Back", showMainMenu))]
class FinishMenu(Menu):
        def __init__(self, text):
                super(FinishMenu, self).__init__(text)
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Try again", startGame)), (MenuItem("Quit", pyglet.app.exit))]
                self.create_menu(menuItems)
class PlatformerController(Action):
        on_ground = True
        MOVE_SPEED = 300
        JUMP_SPEED = 500
        GRAVITY = -1200
        def start(self):
                self.target.velocity = (0, 0)
        def step(self, dt):
                global keyboard, scroller
                if dt > 0.1: # don't do anything while downtime too big
                        return
                vx, vy = self.target.velocity
```

15 **Create an ending for the game.** Even the games that can go on endlessly should have the possibility of losing. Since the level you made in the example with a function has an end, you'll also need to make it possible to win by coming to that end. Otherwise, the player would only jump around on the ice blocks there, which would get boring.

- Inside the platformer controller, after the focus set, get the figure's x and y position. If the y position is less than 0, call the function `finishGame()`(you'll write it later) with `"Game Over"`as an argument. If the x position is bigger than the size of the screen multiplied with 3 (you had set that as level size before).

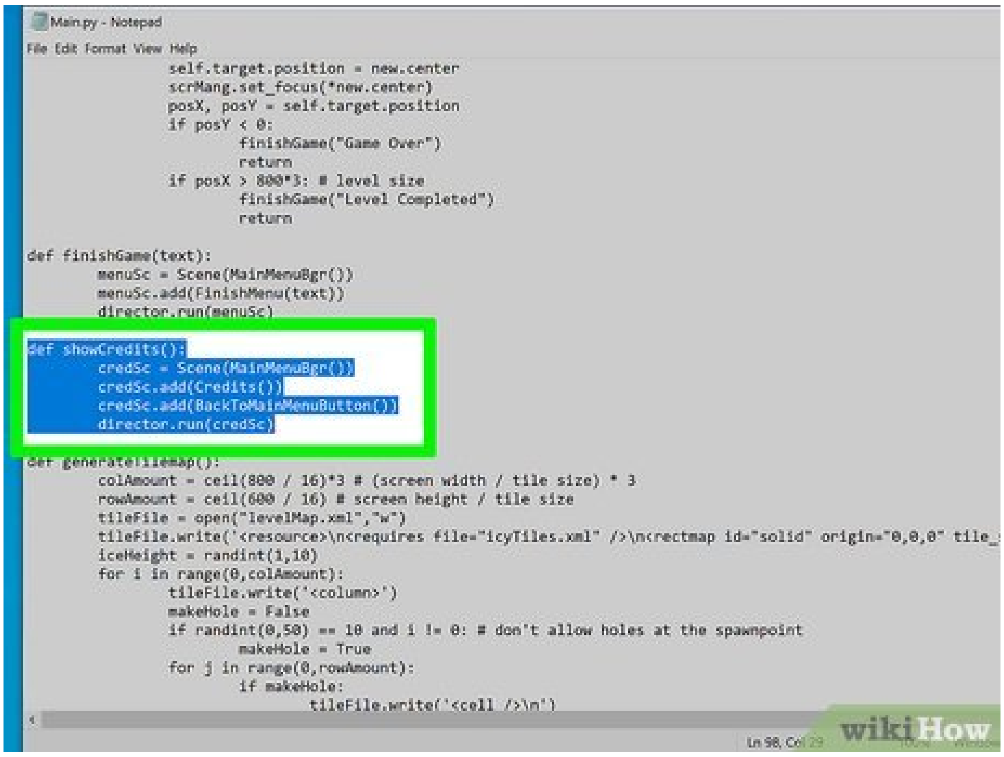
```
posX, posY = self.target.position
if posY < 0:
        finishGame("Game Over")
        return
if posX > 800*3: # level size
        finishGame("Level Completed")
        return
```

- Define a class finishMenu. It should be like the main menu class you defined before, but instead of having an empty string as title, it should use a variable text which the __init__ function takes as argument. The menu items should be labeled "Try again" and "Quit" now, but the functions they call stay the same.

```
class FinishMenu(Menu):
        def __init__(self, text):
                super(FinishMenu, self).__init__(text)
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Try again", startGame)), (MenuItem("Quit", pyglet.
app.exit))]
                self.create_menu(menuItems)
```

- Define the function finishGame(). It should take text as an argument. It should make a scene out of the main menu background, a FinishMenu with the text argument being passed on to this menu. Then it should run this scene.

```
def finishGame(text):
        menuSc = Scene(MainMenuBgr())
        menuSc.add(FinishMenu(text))
        director.run(menuSc)
```



```
Main.py - Notepad
File Edit Format View Help
                self.target.position = new.center
                scrMang.set_focus(*new.center)
                posX, posY = self.target.position
                if posY < 0:
                        finishGame("Game Over")
                        return
                if posX > 800*3: # level size
                        finishGame("Level Completed")
                        return

def finishGame(text):
        menuSc = Scene(MainMenuBgr())
        menuSc.add(FinishMenu(text))
        director.run(menuSc)

def showCredits():
        credSc = Scene(MainMenuBgr())
        credSc.add(Credits())
        credSc.add(BackToMainMenuButton())
        director.run(credSc)

def generateTilemap():
        colAmount = ceil(800 / 16)*3 # (screen width / tile size) * 3
        rowAmount = ceil(600 / 16) # screen height / tile size
        tileFile = open("levelMap.xml","w")
        tileFile.write('<resource>\n<requires file="icyTiles.xml" />\n<rectmap id="solid" origin="0,0,0" tile_
        iceHeight = randint(1,10)
        for i in range(0,colAmount):
                tileFile.write('<column>')
                makeHole = False
                if randint(0,50) == 10 and i != 0: # don't allow holes at the spawnpoint
                        makeHole = True
                for j in range(0,rowAmount):
                        if makeHole:
                                tileFile.write('<cell />\n')
Ln 98, Col 29
```

16 **Add credits.** This is where you get take credit for your awesome code, as well as give credit to anyone else who helped you along the way. If you used an image from another website (with permission), be sure to attribute that image to its creator.

- Create a file `CREDITS` and enter all your credits there, like this:

```
Penguin:
 Kelvin Shadewing, under CC0

Ice block:
 Michał Banas
 digit1024 on opengameart.org
 under CC-BY-SA 3.0
```

- Go back to your Python code and import `Label` from `cocos.text`

- Define a subclass Credits of Layer. In its __init__ function, read the CREDITS file and make a text label at the correct position out of every line in it.

```
class Credits(Layer):
        def __init__(self):
                super(Credits, self).__init__()
                credFile = open("CREDITS","r")
                creds = credFile.read()
                creds = creds.split("\n")
                for i in range(0, len(creds)):
                        credLabel = Label(creds[i], font_size=32, anchor_x="left", anchor_y="to
p")
                        credLabel.position = 25,500-(i+1)*40
                        self.add(credLabel)
```

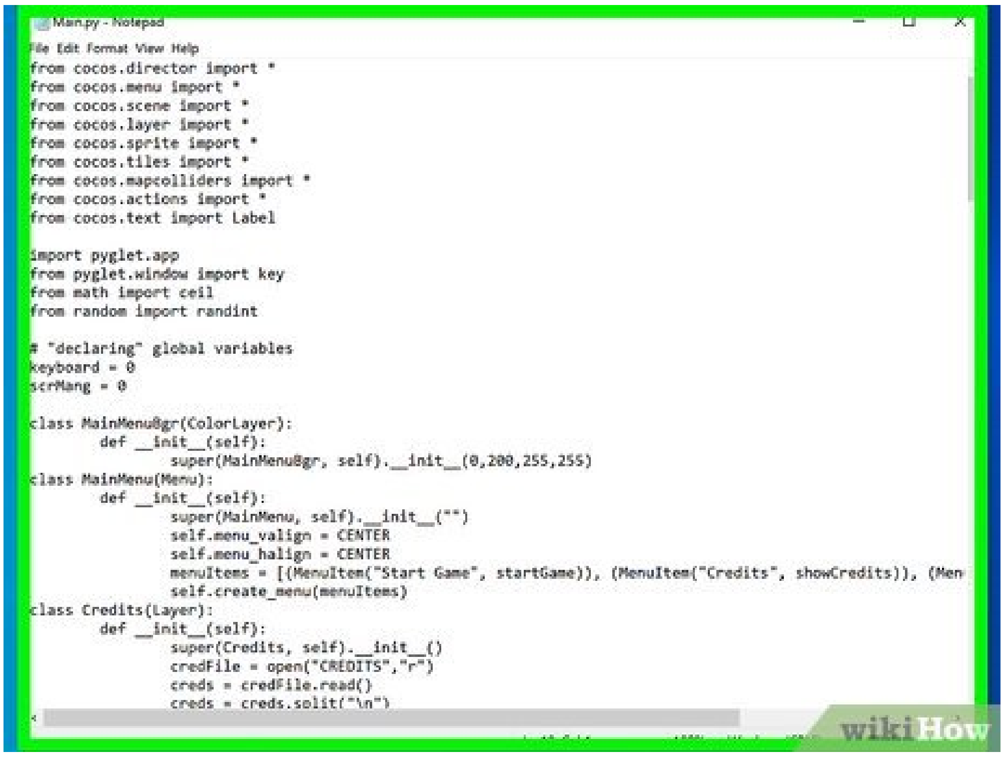
- Go to your main menu class and add a menu item labelled "Credits" that calls the function showCredits when clicked.

- Define a subclass BackToMainMenuButton of Menu. Make this a menu with one item, labelled "Back", that calls the showMainMenu function. This "menu", which is more like a button, should be vertically aligned to the bottom and horizontally to the top.

```
class BackToMainMenuButton(Menu):
        def __init__(self):
                super(BackToMainMenuButton, self).__init__("")
                self.menu_valign = BOTTOM
                self.menu_halign = LEFT
                menuItems = [(MenuItem("Back", showMainMenu))]
                self.create_menu(menuItems)
```

- Define the function showCredits. It should make a scene out of a MainMenuBgr layer and a Credits layer and run that scene.

```
def showCredits():
        credSc = Scene(MainMenuBgr())
        credSc.add(Credits())
        credSc.add(BackToMainMenuButton())
        director.run(credSc)
```



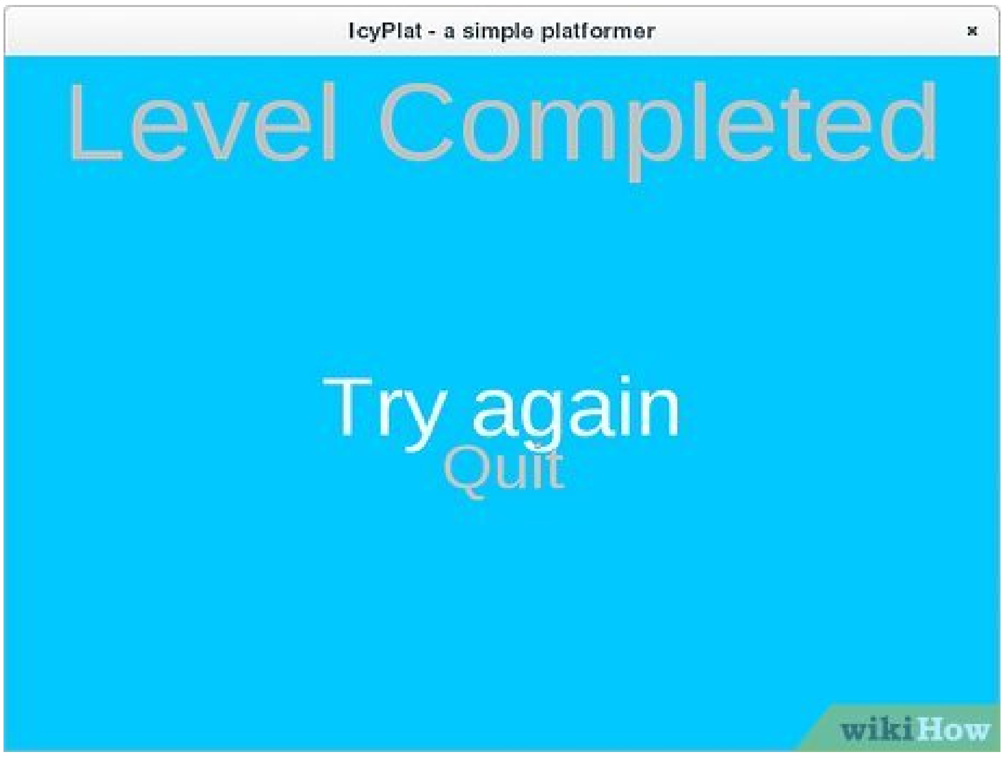
```
from cocos.director import *
from cocos.menu import *
from cocos.scene import *
from cocos.layer import *
from cocos.sprite import *
from cocos.tiles import *
from cocos.mapcolliders import *
from cocos.actions import *
from cocos.text import Label

import pyglet.app
from pyglet.window import key
from math import ceil
from random import randint

# "declaring" global variables
keyboard = 0
scrMang = 0

class MainMenuBgr(ColorLayer):
        def __init__(self):
                super(MainMenuBgr, self).__init__(0,200,255,255)
class MainMenu(Menu):
        def __init__(self):
                super(MainMenu, self).__init__("")
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Start Game", startGame)), (MenuItem("Credits", showCredits)), (Men
                self.create_menu(menuItems)
class Credits(Layer):
        def __init__(self):
                super(Credits, self).__init__()
                credFile = open("CREDITS","r")
                creds = credFile.read()
                creds = creds.split("\n")
```

17 **Check your code.** When you think you finished your code, you should look over all of it again. This can help you notice if something can be optimized, or whether there are some unnecessary lines you forgot to delete. If you followed the example, your entire code should now look as follows:

```
from cocos.director import *
from cocos.menu import *
from cocos.scene import *
from cocos.layer import *
from cocos.sprite import *
from cocos.tiles import *
from cocos.mapcolliders import *
from cocos.actions import *
from cocos.text import Label
from cocos.text import Label

import pyglet.app
from pyglet.window import key
from math import ceil
from random import randint

# "declaring" global variables
keyboard = 0
scrMang = 0

class MainMenuBgr(ColorLayer):
        def __init__(self):
                super(MainMenuBgr, self).__init__(0,200,255,255)
class MainMenu(Menu):
        def __init__(self):
                super(MainMenu, self).__init__("")
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Start Game", startGame)), (MenuItem("Credits", s
howCredits)), (MenuItem("Quit", pyglet.app.exit))]
                self.create_menu(menuItems)
class Credits(Layer):
        def __init__(self):
                super(Credits, self).__init__()
                credFile = open("CREDITS","r")
                creds = credFile.read()
                creds = creds.split("\n")
                for i in range(0, len(creds)):
                        credLabel = Label(creds[i], font_size=32, anchor_x="left", anchor_y="to
p")
                        credLabel.position = 25,500-(i+1)*40
                        self.add(credLabel)
class BackToMainMenuButton(Menu):
        def __init__(self):
                super(BackToMainMenuButton, self).__init__("")
                self.menu_valign = BOTTOM
                self.menu_halign = LEFT
                menuItems = [(MenuItem("Back", showMainMenu))]
                self.create_menu(menuItems)
class FinishMenu(Menu):
        def __init__(self, text):
                super(FinishMenu, self).__init__(text)
                self.menu_valign = CENTER
                self.menu_halign = CENTER
                menuItems = [(MenuItem("Try again", startGame)), (MenuItem("Quit", pyglet.
app.exit))]
                self.create_menu(menuItems)
class PlatformerController(Action):
        global keyboard, scrMang
        on_ground = True
        MOVE_SPEED = 300
        JUMP_SPEED = 500
        GRAVITY = -1200
        def start(self):
                self.target.velocity = (0, 0)
        def step(self, dt):
                global keyboard, scroller
                if dt > 0.1: # don't do anything while downtime too big
                        return
                vx, vy = self.target.velocity
                vx = (keyboard[key.RIGHT] - keyboard[key.LEFT]) * self.MOVE_SPEED
                vy += self.GRAVITY * dt
                if self.on_ground and keyboard[key.SPACE]:
```

```
                vy = self.JUMP_SPEED
            dx = vx * dt
            dy = vy * dt
            last = self.target.get_rect()
            new = last.copy()
            new.x += dx
            new.y += dy
            self.target.velocity = self.target.collision_handler(last, new, vx, vy)
            self.on_ground = (new.y == last.y)
            self.target.position = new.center
            scrMang.set_focus(*new.center)
            posX, posY = self.target.position
            if posY < 0:
                finishGame("Game Over")
                return
            if posX > 800*3: # level size
                finishGame("Level Completed")
                return

def finishGame(text):
    menuSc = Scene(MainMenuBgr())
    menuSc.add(FinishMenu(text))
    director.run(menuSc)

def showCredits():
    credSc = Scene(MainMenuBgr())
    credSc.add(Credits())
    credSc.add(BackToMainMenuButton())
    director.run(credSc)

def generateTilemap():
    colAmount = ceil(800 / 16)*3 # (screen width / tile size) * 3
    rowAmount = ceil(600 / 16) # screen height / tile size
    tileFile = open("levelMap.xml","w")
    tileFile.write('<resource>\n<requires file="icyTiles.xml" />\n<rectmap id="solid" ori
gin="0,0,0" tile_size="16x16">\n')
    iceHeight = randint(1,10)
    for i in range(0,colAmount):
        tileFile.write('<column>')
        makeHole = False
        if randint(0,50) == 10 and i != 0: # don't allow holes at the spawnpoint
            makeHole = True
        for j in range(0,rowAmount):
            if makeHole:
                tileFile.write('<cell />\n')
            else:
                if j <= iceHeight:
                    tileFile.write('<cell tile="ice" />\n')
                else:
                    tileFile.write('<cell />\n')
        iceHeight = randint(iceHeight-5, iceHeight+5)
        if iceHeight < 0: # limit tiles from going too low
            iceHeight = randint(1,5)
        if iceHeight > rowAmount: # limit tiles from going too high
            iceHeight = randint(int(rowAmount/2)-5,int(rowAmount/2)+5)
        tileFile.write('</column>\n')
    tileFile.write('</rectmap>\n<rectmap id="not_solid" origin = "0,0,0" tile_size="16x1
6">\n')
    for i in range(0,colAmount):
        tileFile.write('<column>')
        for j in range(0,rowAmount):
            tileFile.write('<cell tile="sky" />\n')
        tileFile.write('</column>\n')
    tileFile.write('</rectmap>\n</resource>\n')
    tileFile.close()

def startGame():
    global keyboard, scrMang
    generateTilemap()
#
    fig = Sprite('pingu.png')
    fig.position = (8, 250)
    figLayer = ScrollableLayer()
    figLayer.add(fig)
#
    tileLayer = load('levelMap.xml')
    solidTiles = tileLayer['solid']
    nsoliTiles = tileLayer['not_solid']
#
    keyboard = key.KeyStateHandler()
    director.window.push_handlers(keyboard)
#
    fig.do(PlatformerController())
    mapcollider = RectMapCollider(velocity_on_bump='slide')
    fig.collision_handler = make_collision_handler(mapcollider, solidTiles)
#
    scrMang = ScrollingManager()
    scrMang.add(nsoliTiles,z=-1)
    scrMang.add(solidTiles,z=0)
    scrMang.add(figLayer,z=1)
#
    gameSc = Scene(scrMang)
    director.run(gameSc)

def showMainMenu():
    menuSc = Scene(MainMenuBgr())
    menuSc.add(MainMenu())
    director.run(menuSc)

window = director.init(caption="IcyPlat - a simple platformer", resizable=True)
showMainMenu()
```

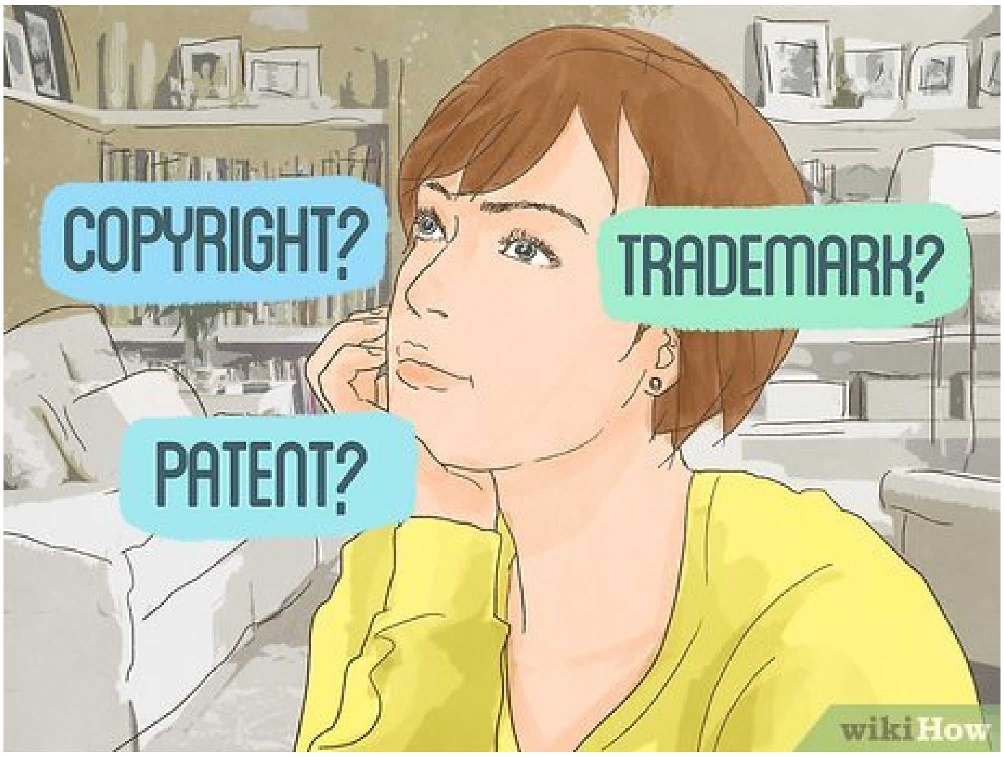
- That are 168 lines totally, and 152 lines if you only count the code. This make seem like much, but for such a complex game, this actually is a small amount.



18 **Finished.** Now test the game. When you program something, you have to check whether it works whenever you have implemented something new. Also, you might like to play the game you wrote for some time.

Part 3 of 3: Publishing a Game

1 **Write down the dependencies.** Anyone who uses another computer will not have the same software and libraries installed as you. So, you'll need to make sure everyone who installs your game knows exactly what they'll need to run it. You don't have to write down all dependencies of all dependencies of all dependencies and so on, but you should at least write the dependencies of your packages and their dependencies.



2 **Make sure you have permission to use all media.** This applies to all graphics, including 3D models, music, dialogue, music, libraries, and frameworks you used for your game. Anything you didn't write yourself.

- Often there are some conditions, like having to credit the author or share modifications of the media under the same license. Sometimes you'll be able to use graphics without attributing the creators as long as you don't charge for the game. If you have to credit the author, do it in a well-visible place, like a "Credits" tab in your game.
- There is also media with copyright claimed and no license specified, sometimes with some text like "All rights reserved". If that's the case, you must get explicit permission from the author before including it in your game.
- Libraries are usually released under licenses that allow them to be used as library. A notable exception is the GPL without linking exception: Such a license only allows to use it in a program with certain licenses. And you should always read at least the basic points of the license to make sure whatever you're doing with the media or library is allowed.

Warning: Using media or libraries in a way that the license doesn't permit in a game that you publish can get you into serious legal trouble. So either ask the author or avoid the piece of media altogether if you are unsure about whether your usage is allowed.

3 **Decide on the conditions you want to publish your game on.** Will you be selling your game? Do you want to allow others to use your images and ideas? While you have to be careful about the media you use in your project, you usually can decide on how you want to allow *others* to use *your* game. You can use a Creative Commons CC0 license to release your game in the public domain.[1] . To allow distribution and modification under some conditions while retaining some rights, try the Gnu General Public License (GPL) or the Berkeley Software Distribution (BSD) license. Or, you could make your software proprietary, meaning that nobody is allowed to distribute or modify it without your permission.

- Although it is possible to make money by selling games, it is unlikely that people will buy your first game that usually has few features and nothing special. Also, if a free program doesn't work, people who downloaded it will just be disappointed. If they paid for it, however, they'll demand their money back, causing more problems for both you and the users. So consider making your first few programs available for free.

4 **Decide how you want to publish your game.** Every method has some advantages and disadvantages, so you have to decide yourself.

- **Publishing it on a website:** If you have a website, you can upload your game to make it available for download. Make sure to provide clear instructions on how to install the software, as well as all required dependencies. The disadvantage of this is that players will have to install dependencies manually, which might be difficult for some people.
- **Making a package for a package manager:** There are different package managers, like apt, Yum, and Homebrew, that make it easy for people to install apps in Linux and Linux-based environments. They all have different package formats. The good thing about packages is that they automatically install all dependencies (if you configure them correctly). So the player only has to install your package and can then play the game. The problem is that there are many different package managers on different platforms, so you will have to put some work into providing packages for all the most common ones.

5 **Direct attention to your program.** Consider uploading your program to a major package repository, like the ones Ubuntu and Debian maintain, to allow for easy installs. Also, post in appropriate forums, like the projects section of GameDev or a part of tigSource. But don't be disappointed if your first games don't become famous. If you have an idea that many people like it, your game can become well-known.

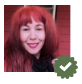
Tips

- Be patient and willing to learn. Programming can be frustrating at times!
- If you wonder how something is done in another game, and the game is open-source, you can look at its source code.
- When looking for media, try to find content that's in the public domain. Search for "Creative Commons" or "Public Domain" images and music, and use websites like https://opengameart.org or https://publicdomainpictures.net.
- Don't copy major chunks of code without checking the license. It is often prohibited, and if not, usually requires attribution.
- Don't make spam or post in inappropriate places when promoting your game. This is likely to get you blocked from the page, is simply annoying, and will damage your reputation.

References

1. ↑ https://creativecommons.org/share-your-work/public-domain/cc0/

About This Article



Co-authored by:
Nicole Levine, MFA
wikiHow Technology Writer

This article was co-authored by Nicole Levine, MFA. Nicole Levine is a Technology Writer and Editor for wikiHow. She has more than 20 years of experience creating technical documentation and leading support teams at major web hosting and software companies. Nicole also holds an MFA in Creative Writing from Portland State University and teaches composition, fiction-writing, and zine-making at various institutions. This article has been viewed 7,321 times.

Co-authors: **18**
Updated: **June 23, 2021**
Views: **7,321**

Categories: Programming | Video Game Creation

https://www.wikihow.com/Program-Computer-Games