Intro:

Hi, I’m Richard Kelly, creator of Ninja’s Vs. Pirates Vs. Robots, a multiplayer turn based strategy game created using python, pygame, and livewares.

Some of my favorite games are games like Fire Emblem, Advanced Wars, Civilization, or Final Fantasy Tactics. So I set out to make a game with similar concepts. In the game, players take turns controlling an army of ninjas, pirates, and robots and attempt to eliminate their opponent’s army. The basics are easy, Ninjas beat Pirates, Pirates beat Robots, and Robots beat Ninjas in a rock paper scissors style of interaction. However, an element of randomness is introduced, and sometimes the Ninja’s might lose to the Pirates and so on. You never really know what is going to happen. After all, if there wasn’t a random element, you almost might as well be playing a board game.

The game is a hotseat game, where two players switch off taking turns at the same computer. The map, the number of units, where the units are, and what type of units they are can be defined before each game in files called level1.map and level1.units. These files let you play the game how you want, and allow you to create your own story. Are you battling it out on Robot-Pirate Island? Or maybe hordes of Ninjas are fighting against just a few Robots battling for their Robot lives. By editing these files. the players can define the game as they see fit.

I hope you all enjoy the game, and have as much fun playing with it, and picking it apart, as I did writing it.

Composite Background:

I’m Richard Kelly, creator of Ninjas Vs Pirates Vs Robots.

One feature of Ninja’s Vs. Pirates Vs. Robots is a composited background made up of image tiles. What this means is, rather than creating a background image in an image manipulation program such as Photoshop, Paint, or GIMP a set of smaller images are put together at run time in order to create the background image seen in game.

The first step for this is to define how you want your images put together. I decided to use a file called a CSV, or comma separated values file, because python has built in support for reading these kinds of files in as lists.

As you can see, the file consists of file names separated by commas for each column, and a new line signifies each row. To read this, we have our board class, located in Board.py. Board.py uses the function loadBack to read in each row of the csv as a list, remembers how wide the board is and then increments the height of the board by one. It then adds the list of files from the csv to its own list.

To mash all of these images together we use a Pygame class called a Surface. Many places throughout the livewires library it asks for images. While this may seem like it needs an image file, what it actually wants is a pygame Surface. A Surface is created whenever we call games.load\_image() on an image location. This means, that we can pass any surfaces we want to these functions. This is useful, because Pygame allows us to make a surface of surfaces by using a function called blit. All we have to do is pass blit the surface we want to use, and the coordinates of the where we want it on the larger surface.

We use our for loop to iterate based on the remembered dimensions of the board from our board class, and then set our games.screen.background equal to the surface we created.

For some practice, try editing the CSV file. The program as it is written is expecting images that are 60 by 60 pixels and located in the img/ folder, so, if you want to use images other than the ones I used, try to keep to those requirements.

Once you’ve played with the CSV file, you may want to edit the for loop that places the tiles. Try and gain a better understanding of how it works.

Inheritance:

Hi everyone, I’m Richard Kelly, creator of Ninjas Vs. Pirates Vs. Robots. Right now I’d like to talk about an important concept in Object Oriented Programming called Inheritance. This video assumes you already know a little about object oriented programming, such as what an object is and what its member functions are.

Now, all classes in python by definition use inheritance, most just inherit from the “object” class. When you declare a class in python it may be declared like this

(class MyClass:)

or like this

(class MyClass(object):

both mean the same thing.

Let’s stop right now and define inheritance. Inheritance is when one object is based on another object. It gains all of the same member functions and variables. It is a way of reusing code and creates a hierarchy. Inheritance is used to add functionality that the original class does not have

For things that inherit from object, this doesn’t really mean anything. So let’s look at an example of where inheritance was used in Ninjas Vs Pirates Vs. Robots.

This is the Piece class, it defines most of the background functions of the pieces that move across the board in NPR, as well as defining most of the variables needed to keep track of a piece. Here we see the Ninja class. It inherits from Piece. Ninja has its own init function, which calls Piece’s init function by calling super(). Ninja also has a function that Piece does not have, the attack function.

Robot and Pirate are very similar to Ninja. In fact, it may be possible as extra practice, to create a class that inherits from Piece and has an attack function, and then have Ninja Pirate and Robot inherit from that class.

One of the most common uses for inheritance while using Pygame and Livewires is to inherit from the sprite class so that the update function may be edited. This can be seen several times in game.py. When you inherit from a class and create a function with the same name as one that the parent has, it overrides that function and changes its behavior. Update normally does nothing, but once overridden it is one of the most important functions in a livewires program.

One final thing about inheritance. So far, I’ve talked about inheriting from one parent class, but what happens if you want your class to have the same functions as TWO or even more parents? Some programming languages, such as Java, do not allow this sort of Multiple Inheritance. However, Python does. I will caution you that multiple inheritance can make life tricky. What happens if both parents have a function called getX? Whose gets called? I will leave multiple inheritance, how to do it, and some of its pitfalls as an exercise for you to explore.

Levelling Up

I’m Richard Kelly, creator or Ninjas Vs Pirates Vs Robots, and this is the third tutorial video about my game. In the last video I talked about the object oriented programming concept of inheritance, and in this video I’d like to talk about something more specific to my game: How to have characters keep track of experience and level up. In my last video I introduced the Piece class, the Ninja class, the Pirate class, and the Robot class. We need these if we are going to track stats for individual units on the field.

The basics for leveling up vary from game to game, but boil down to a simple concept: whenever a unit does something it may gain experience, and, if it has enough experience, it will level up. So, we need two things in our piece class. An experience variable, along with the necessary getters and setters, and a level variable and its getters and setters.

If we go to our attack functions, we see that whenever a unit wins a battle it gains experience, and, whenever a unit gains experience, we check to see if it has hit a threshold to gain a level. If it has, we increment the level, and increase its health. If you look at the amount of damage the unit does, we base that partially on the level of the unit.

It can’t be that easy though, can it?

It basically is. With a minor drawback. While the backend of our program knows that the unit has gained a level, we still need to display that to the user. So how do we do that?

In game.py we look at the Unit sprite, and see its battle function. Before a battle happens, it remembers the level of both units involved. After the battle, it checks again. If the level has changed, it displays the message that a level up has occurred.

On the side bars, level is tracked by setting it during every update. Whenever the level changes, we will immediately see a change on the appropriate units listing.

In your own game, you may want to have level be tied to various stats, such as strength, or speed, or defense, and then have those be tied to how your unit acts in combat, such as how games like Pokemon, Final Fantasy, or Mass Effect work.

For some practice, try changing the amount of experience given for winning a battle on offense or defense. And for some more advanced practice, give a bonus in experience if your enemy has had their life brought to or below 0.