ECE 122: Introduction to Programming for ECE- Spring 2022

Project 3: Fun with Games (OOP and Graphics)

Due Date: **See website, class policy and moodle for submission**

Description

The goal of this project is to practice OOP using multiple classes, instance variables, methods (instance and static), inheritance, and some encapsulation. You will also operate Graphics to develop multiple games.

Your project zip file must include multiples files:

1. The classes: `Counter.py`, `Dot.py`, `Explosion.py`, `Missile.py`, `Alien.py` `SpaceShip.py`
2. The application files: `fireworks.py`, `game1.py` and `game2.py`
3. Multiple png files for aliens, spaceship and background pictures.

the figure below represents the inter-relationships between all classes and applications:

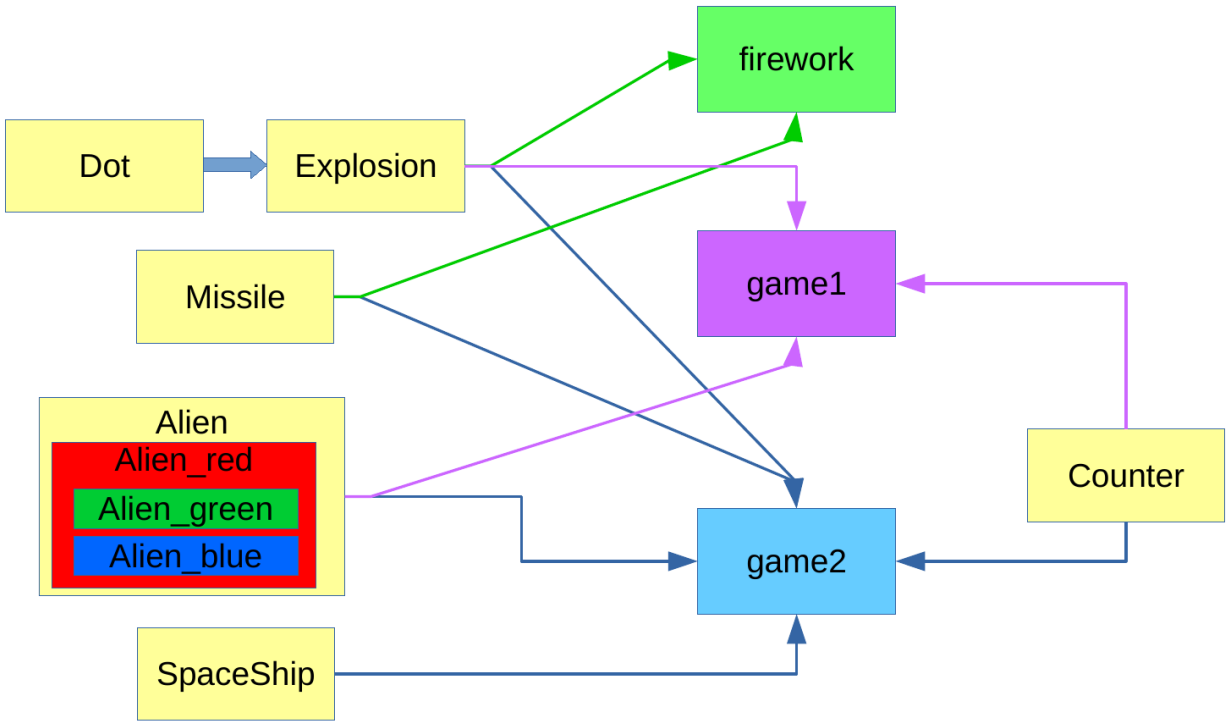

Figure 1: `Relationships between classes (yellow boxes) and Applications.`

Each class can be implemented and tested separately (they have their own main method).

Submission/Grading Proposal

You will regroup all your files (including png pictures) into one zip file at the time of submission (*do not* create a new folder). this project will be graded out of 100 points:

1. Your program should implement all basic functionality/Tasks and run correctly (100 points).
2. Overall programming style: program should have proper identification, and comments (up to -5 points if not).
3. You can do the project alone or by group of 2 max. If you do it in team, then submit only **one zip** file (anyone of the two). You will add your two names in the header of the file `Dot.py`.

How to start

All the classes can be implemented in pretty much any orders, however I would suggest to follow the order of the tasks that are presented below. There are a lot of things to complete, so if you get stuck somewhere, you can just move on. For the application `fireworks` you need to have the classes Dot, Explosion and Missiles implemented. For `game1` you need to have the classes Explosion, Alien and Counter. For `game2`, all classes must be implemented.

Class Dot- [10pts]

When running `Dot.py`, the main function of the program is executed. A rectangular black window 800x1000 should appear. If you left click on your mouse somewhere in that window, you should see a colored dot that appear (you can repeat that action to get multiple colored dots). In addition, the (x, y) tkinter coordinates and the color of each dot is also printed. Figure 2 shows a snapshot example of what could happen after multiple clicks.

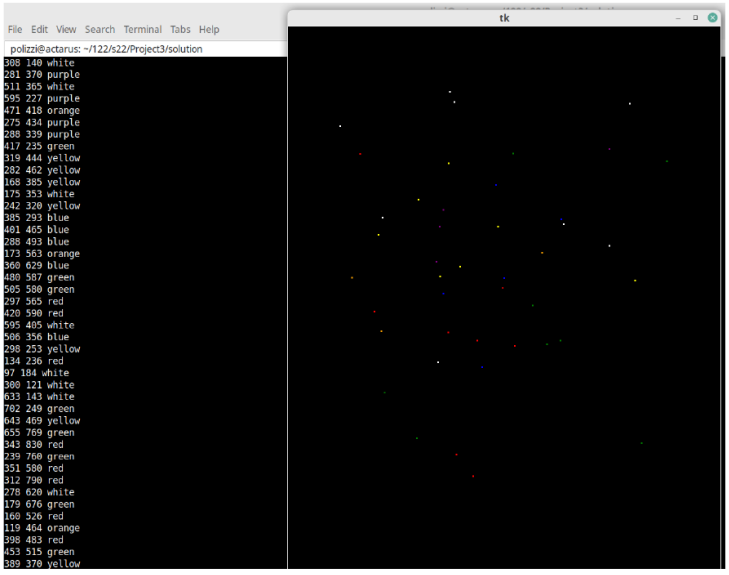

Figure 2: `Dot.py`. With a left mouse click, a dot appears with a random color.

How to proceed?

- The main function is provided. It includes the creation of Tk window and the canvas. It also includes a binding instruction between a particular event (left mouse click here) and a particular action. In general, a binding can be set up as follows:

  `root.bind("<Event>",lambda e:action())`
  where "Event" represents a particular event such as `Button-1` for left click, or `Left` for left arrow, `Right` for right arrow, etc. Also `action` stands for a particular method or function that will be called if the event happens. We note that `lambda` is a Tkinter keyword used to create a link between a Tkinter event `e` and a callback function. In the our particular case, we have:

  `root.bind("<Button-1>",lambda e:Dot(canvas,e.x,e.y,"rainbow",True))`
  It means that after a left click, a Dot object will be instantiated (the method `__init__` of the class Dot will be called). Among the arguments, we have the position x and y automatically returned by the tkinter event (where you click inside the window).

- You need to implement the constructor of the class `Dot`. It should accept five arguments: the canvas, the x and y position of the dot to draw, the color of the dot, and a Boolean argument that acts on the display message. The latter must be set to `False` by default, which means that no information about the position of the dot and its color is displayed (no print). You also need to consider the following:

  - the color of the dot is specified in the call to the function. For example, you could call the constructor using the color `blue`, `yellow`, etc. Here we call it using our customized color `rainbow` which means that a color must be selected at random in the constructor between red, green, blue, yellow, white, orange, purple. Hint: a call to the function `random.choice(items)` of the module `random` will return a random item from the list "items".
  - the dot which is plotted must be an oval centered at the x, y provided coordinates and of radius 1 pixel (so 2 pixels diameter). The filling and boundary colors of the dot must be of the same color.

Class Explosion- [20pts]

The class `Explosion` is using the class `Dot` to generate a set of random dots within circle layers that gradually expands from a x,y position toward a maximum radius, before disappearing (i.e. mimicking an explosion). If you left click on your mouse somewhere in that window, you should see a multi-colored explosion that appear. It is also displaying the active status of all the explosions (True or False) before a new explosion is clicked (explained further below). Figure 2 shows a snapshot example of what could happen after multiple clicks.
How to proceed?

- The main method is provided. In addition to the tkinter, window and canvas (seen in the `Dot` class), it includes an empty list of explosions that will start to be filled up with object of type `Explosion` as soon as you left click on the mouse. This is achieved by calling the static method `add_explosion` in the binding statement. The main method also includes a while loop with a time sleep instruction of 0.03s that will pause the animation each time the method `next` for a particular explosion is called. The `next` method represents the next time

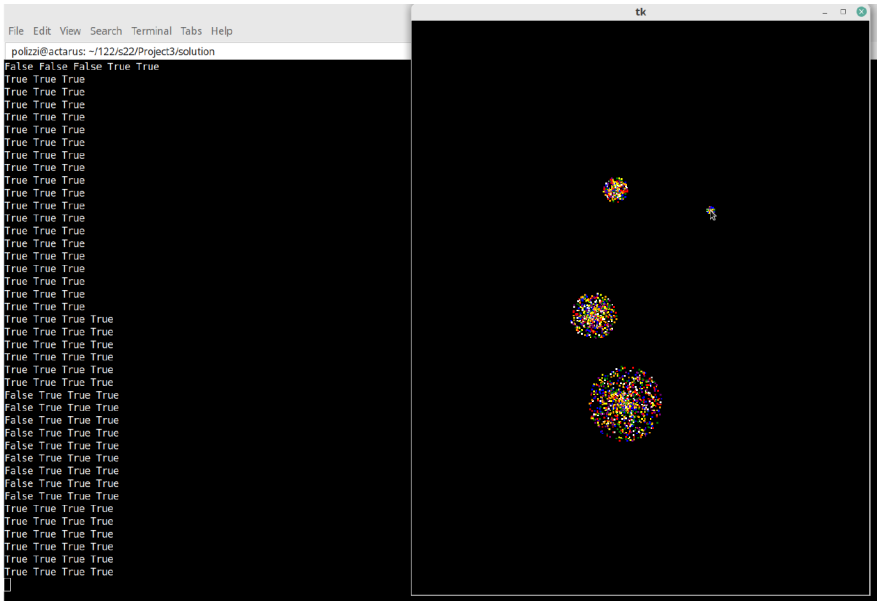

Figure 3: `Explosion.py`. With a left mouse click, a explosion appears.

iteration for the explosion. Indeed, the explosion is growing layer of dots by layer of dots and the next iteration represents the next layer.

- You need to implement the constructor of the class `Explosion`. It must have four arguments: the canvas, the max radius of the explosion that you will set to 80 pixels by default, and the color that you will set to `rainbow` by default. The constructor should contain other instance variables such the number of random dots by layers (set to 15), and a variable to store the list of all dots generated (empty list to start with). Finally the constructor should contain a **private** variable named `active` set to False in the constructor. All other variables can be public.

- You need to implement a method `activate` that uses two arguments: the x,y coordinates of the explosion. You will here create three new instance variables: x,y, and r the current radius of the explosion (set at 0 to start with). Obviously the private variable `active` should be set to True here.

- You need to implement a method `is_active` that return a Boolean (the value of the private variable `active`).

- The `next` method represents the "next time iteration". If the private variable `active` is True, you can increment the current radius by 1 (we are growing the layers 1 pixel at the time). For the new layer, you need to generate 15 random dots (max number of dots was set in a constructor). Here it is useful to use a polar coordinate system where you could generate angles at random to obtain the x,y coordinates for each new dot (you also know the center coordinates of the circle which was saved in the `activate` method). Hint1: a call to the function `random.randint(0,359)` of the module `random` will return an int at random from 0 to 359. Hint2: the cosine and sine functions are available in the `math` module..do not forget that the angles are in radians.
  Once a dot is generated it can be appended to the list of dots (defined in the constructor).

You also need to test if the radius becomes greater than the max radius set in the constructor. If so, you will call the method `deactivate`

- You need to implement the method `deactivate` that will delete all the dot objects in your list of dots. You can achieved this using the delete method of the canvas object (look it up). Obviously the private variable `active` should also be set to False.

- You need to implement the static method `add_explosion` which is the function called by the binding instruction (once you click left). It must have six arguments: the canvas, a list of explosions, the center (x,y) coordinates of the new explosion, the max size of the explosion radius set to 80 pixels by default, and the color of the explosion set to "rainbow" by default. Inside this function you must instantiate a new explosion, activate it, and add it to the list of the explosions.

  At this point, the code should work fine. By clicking on a new position in the window you would be able to generate a new explosion. In addition, the `active` status of each explosion should be displayed. The explosion active flag originally set to "True", should convert to "False" as soon as the explosion is done. You need to test the program before proceeding to the final stage below.

- Final stage: at this point if you keep clicking, the list of explosion will keep growing in size. This is not very clever since you are keeping in memory explosions that may not be active anymore (the list will grow indefinitely as well). As a result, before appending a new explosion in the method `add_explosion`, it would make more sense to clean up the list of current explosions by deleting (removing) the explosions (item in the list) that are not longer active. Hint: once you find the index of the item to remove from the list, you can use the method `pop`.
  One way to test this feature (necessary but not sufficient), you can start by clicking a bit everywhere multiples times, wait a certain time for all the active status to turned to False, then click one more time. The entire list of False, should become a single True. If you do not succeed to implement this feature, ask a TA to send you the method `add_explosion`, it will cost you -3pts but it is an important method/feature that can be used as a template to implement other similar methods in this project (for the other classes).

Class Missile- [15pts]

The class `Missile` is sending rectangular projectiles from the bottom of the canvas and at a random x position, toward a certain height (chosen at random). There is no binding event (such as mouse click here), a new missile is automatically sent every 0.5s. Similarly to the explosions (previous task) the active status of all the missiles (True or False) is displayed. Figure 4 shows a snapshot example of what could happen after some time.
How to proceed?

- The main method must be completed. Take inspiration of the main method for `Explosion` to do it. The main difference here is that there is no tkinter binding happening, and that you need to keep track of the time in the while loop. Every 0.5s, you will call the static method `add_missile` that will keep appending a new object of type `Missile` to a list of missiles (very similar to the static method in the class `Explosion`).

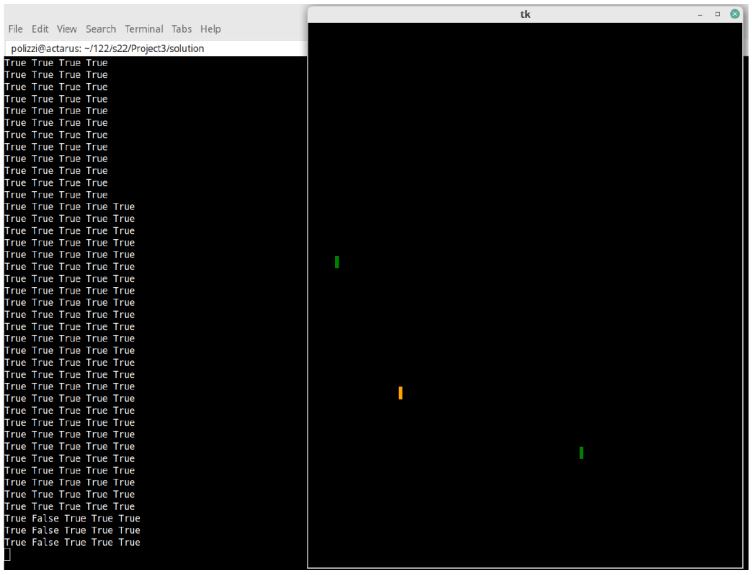

Figure 4: `Missile.py`. A new missile appears every 0.5s and it reaches a certain height. Missiles may have different speed, color, and limit height.

- The `add_missile` method should have seven arguments: the canvas, the list of missiles, the (x,y) starting coordinates of the new missile, the ceiling height of the new missile (in the tkinter coordinate format, so 0 means top of the canvas), the pixel increment of the missile by time iteration set to 5 as default (which represents the speed the missile), and the color of the new missile set to "orange" as default.
- For the class `Missile` you need to implement:
  - the constructor that uses six arguments: the canvas, the ceiling height (in tkinter format) set at 0 by default, the pixel increment set at 5 by default, the color set at "orange" by default, the width of the missile set at 8 by default, and the height of the missile set at 25 by default. You will also initialize the private variable `active` as False.
  - the method `activate` using (x,y) coordinates as arguments. It must set and plot the rectangle missile. We note that x represents the center of the rectangle missile, while y represents here the bottom of the rectangle missile. Also, private variable `active` is set to True.
  - the method `deactivate` which delete and deactivate the missile.
  - the method `is_active` which returns the active status of the missile.
  - the method `next` which, if the missile is active, moves the missile up by 5 pixels (increment value defined in the constructor). You can use the method `move` of the object canvas to do this (seen in class lec 3.4). You also need to test if the new y position of the missile becomes smaller (in tkinter coordinate) than your max ceiling. If so, you will call the method `deactivate`.
- When calling the `add_missile` in the main method, you will first generate a random x coordinate (along the width of the canvas), a random ceiling max (along the height of the canvas), a random pixel increment (speed) chosen between 2 and 7, and a random color chosen among blue, yellow, green, purple, red, orange. All of these random values will be used as arguments for `add_missile`.

Application fireworks- [20pts]

It uses the classes `Explosion` and `Missiles` to generate random fireworks. Figure 5 shows a snapshot example of what could happen after some time.

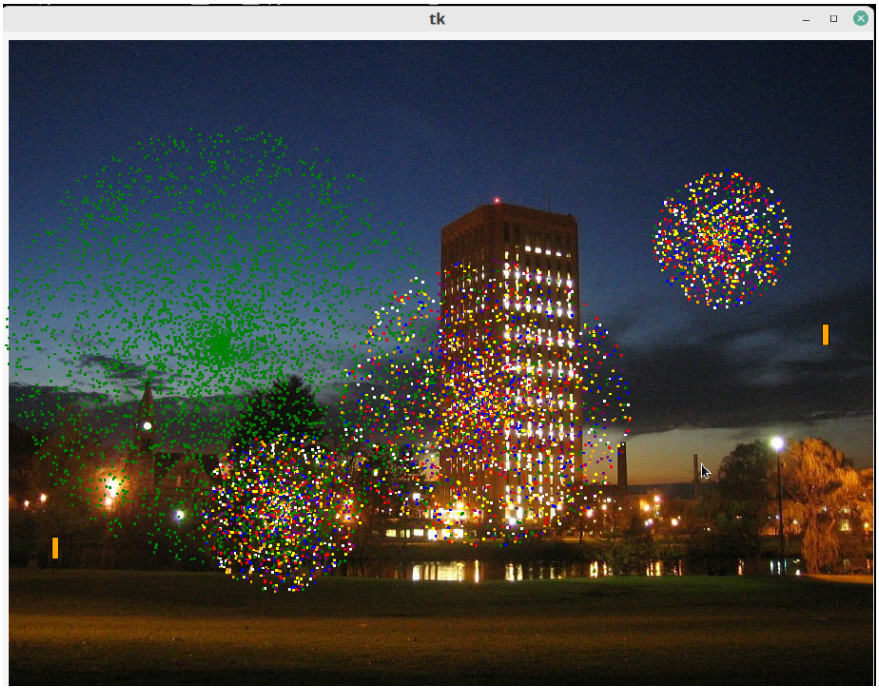

Figure 5: `fireworks.py`. A beautiful firework at UMass!!

How to proceed?

- Complete the main program. You can use the (night) picture of your choice (but be sure to include your png file in the zip document!!).
- Here we need to manage a list of explosions and a list of missiles.
- A new missile is launched every 0.5s. The x position of the missile is random and the max ceiling could be between h//4 and 3*h//4 with h, height of the canvas.
- Before calling the `next` method for a particular missile, you may want to check its active status. If its active status changes from active to not active after you are calling `next`, it means that the missile has reached its max ceiling and a new explosion can be added to the list of explosions at those (x,y) coordinates.
- Every new explosion should have a maximum radius chosen at random between 100 and 300, and a color chosen at random between rainbow, blue, etc...up to you.. (I use rainbow multiple times in my list of colors to increase its chances to get chosen).

Class Alien- [30pts]+[5pts-bonus]

This class is used to generate different types of aliens that are dropping at random from the top of the canvas toward the bottom. There is a parent class `Alien`, its child class `Alien_red` which, in turn, is the parent of the classes `Alien_green` and `Alien_blue`. All the alien subclasses inherit the `Alien` methods. The alien's motion behave differently and its shape is different depending the class it belongs to. In addition, if you left click, a message will display the coordinate of your click and some info about if you managed to hit or miss the alien. Figure 6 shows snapshot examples of what could happen after some time for various types of aliens.

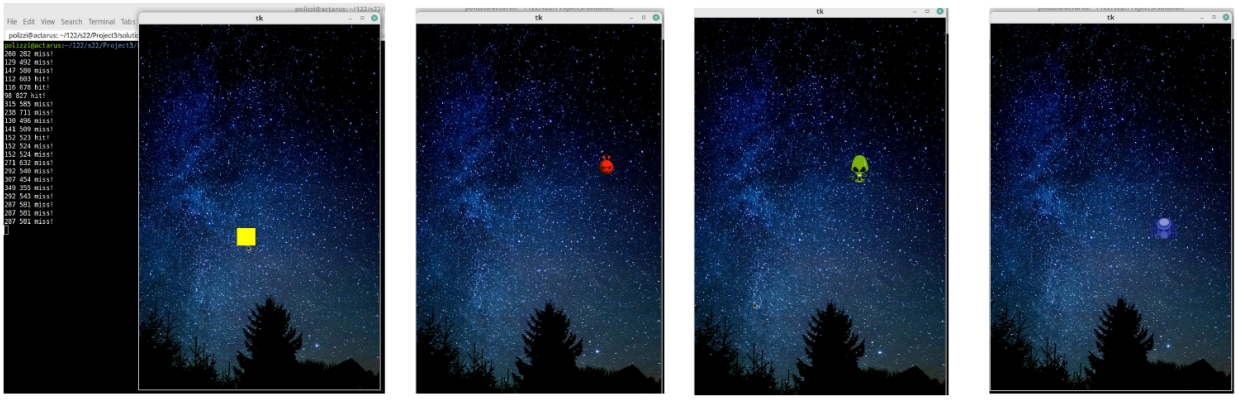

Figure 6: `Alien.py`. Only in your worst nightmares.

How to proceed?

- The main method is provided and you can uncomment or comment the type of alien you want to instantiate for testing.
- The function `shoot` is also provided. It represents the action linked with the left click binding.
- You first need to implement the parent class `Alien` (the nasty yellow square). It contains the following methods:
  - the constructor that uses six arguments: the canvas, the pixel increment (set at 4 by default), the color (set at yellow by default), the width and height of the alien (both set at 50 pixels by default), and the intrinsic point value of the alien that will be used for gaming (set at 1 by default). You will also initialize the **protected** variable `active` as False.
  - the method `activate` where you will set up the (x,y) coordinates (center of the alien) as new instance variables. The x position is chosen at random along the width of the canvas while y is set at the top of the canvas (be mindful of the height of the alien when setting y). Hint: the width of the canvas can be retrieved using the method: `winfo_width()` of the canvas object. Do not forget to instantiate and activate the alien as the yellow square centered at x,y. Remark: the (x,y) coordinate could be set to None by default as input argument of the method (that provide more flexibility for later general usage if you want to specify your own coordinates).
  - the methods `is_active` and `deactivate`.

  - the method `next` that will move the alien down by the pixel increment (if it is active) until it reaches the bottom of the canvas (which value can be retrieved using the method:`winfo_height()` of the canvas object). if so, the alien is deactivated.
  - a method `is_shot` that uses the coordinates x_0, y_0 of a given hit as argument. If these coordinates end up inside the alien, it means that it has been shot and you must return True (False otherwise).

- Next implement the child class `Alien_red`. Like the yellow square, this red alien is also dropping from the sky. Part of the constructor is provided for you (so you would know how to read an image file using `PhotoImage` and create an image object). You need to complete the constructor by calling the **super** operator. Here you will also use a pixel increment of 4, the new width and height of the alien image, the color "red", and a new intrinsic alien point value of 2 (we use the color and the intrinsic value later for gaming).
  As you know, all the methods of the class `Alien` are inherited. However, you do have to override the method `activate`. Use the same choice than for the `Alien` class for selecting the (x,y) coordinates. Here you need to instantiate the red alien using the method `create_image` of the canvas (see lec 3.4) that you can anchor using the option `CENTER` rather than NW, SW, etc. (as a result, the center of the alien image will be at x,y).

- Next implement the grand-child class `Alien_green` (child of `Alien_red`). This alien also uses a pixel increment of 4, the color "green", and a new intrinsic alien point value of 4. To implement the constructor, you will need to use a call to the grandparent constructor (using `Alien.__init__(self,...)`. Since this alien is using the same method `activate` than the red alien, you just need to override the method `next`. Indeed, this green alien is also dropping from the sky but it keeps wiggling at random horizontally at the same time. In addition to moving it down by the pixel increment, you will move it horizontally by an integer chosen at random between -5 and 5 pixels.

- Next implement the grand-child class `Alien_blue` (another child of `Alien_red`). This alien also uses a pixel increment of 4, the color "blue"', and a new intrinsic alien point value of 3. To implement the constructor, you will need to use a call to the grandparent constructor (using `Alien.__init__(self,...)`. Since this alien is using the same method `activate` than the red alien, you just need to override the method `next`. This blue alien is moving diagonally with a angle chosen at random between (-160,-20) to start with (you can assign a new instance variable for this angle θ in radian in the constructor). This alien is also bouncing at the boundaries. If its edge touches the left boundary, its angle becomes $\pi - \theta$, if it touches the right boundary it becomes $-\pi - \theta$.

- Bonus [5pts]- You can do it once your are done with this project. Write your own child class that you will call `Alien_mine` using some original move for the alien. You can use your own png picture (but include it with the zip file).

Counter- [5pts]

This class generates a counter at the top right of the screen that you will be able to increment/decrement by 1 using the left or right arrows of your keyboard. Figure 7 shows a snapshot example of what should happen as soon as you execute the code.

Figure 7: `Counter.py`. Zoom at the top right.

How to proceed?

- Write the main code
- Write the class which contains only the constructor and the method `increment`. The constructor has two arguments: the canvas and the initial value of the counter (set at 0 by default). Use the method `create_text` of the canvas to generate the text. I used 70 pixel from the right and 20 from the top to place my text, as well as 25 size Courier for font (and orange color). Feel free to use whatever you want.
- The method `increment` has one value that represents the increment (either positive or negative). This method should be "binded" with the left and right arrows in your main code with the +1 or -1 value as input (using the `Left` and `Right` events). Inside this method, you can use the method `itemconfig` of the canvas to modify the value of the test to be displayed.
- When you instantiate the counter in your main code, start with an initial value of 10.

Application game1- [20pts]

Time to implement our first game which regroups features from the class `Alien`, `Explosion` and `Counter`. When you start the game, the counter is set to 10 which represents the number of your ammunition. Aliens (red, blue or green) are coming from the top every 0.5s. If you hit the alien by left clicking on it, it will explode (explosion red, blue or green depending on the alien color) and you will gain some more ammunition (the corresponding alien point values). If you miss it, the explosion will be white and you will loose 3 ammunition point. If the number of ammunition falls to zero or below zero, it is game over. A "GAME OVER" message will appear in the middle of the screen and the game will stop. Figure 8 shows multiple snapshot examples of what could happen after some time.

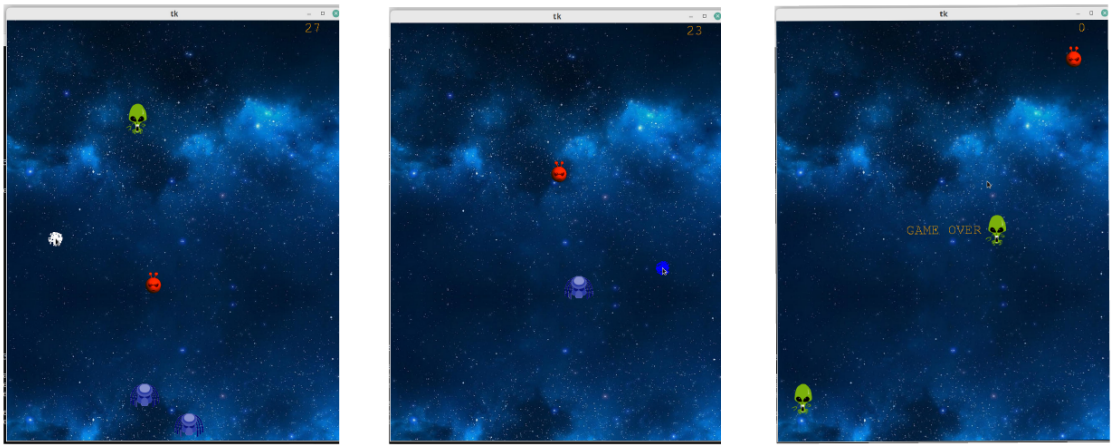

Figure 8: `game1.py`. Having fun chasing aliens.

How to proceed?

- Complete the main program. You can use the background picture of your choice (but be sure to include your png file in the zip document!!).
- Here we need to manage a list of explosions, a list of aliens, and the counter object (initialized to the value 10).
- A new alien appears every 0.5s. You will need to add a static method `add_alien` in the `Alien` class. The method has two arguments, the canvas and the list of aliens. A new alien is then chosen at random between the type `Alien_red`, `Alien_blue`, `Alien_green` and your own `Alien_mine` if you wish (and if you did it for bonus point). Hint: You can instantiate an alien for each of these classes creating a list, and select your new alien at random among that list. You will then need to activate your new alien, clean the list of non-active aliens (like you did for `add_missile` and `add_explosion`, and append it to your list of aliens.
- When you left click, you will call the function `shoot` that you will have to implement. It has six arguments: the canvas, the list of aliens and explosions, the counter object, the coordinated x and y of your click event. This function must accomplish three things:
  - scan the list of aliens. If an alien is both active and is being shot (use the `is_shot` method here), the counter will be incremented by the intrinsic value of each alien (that

was defined in your various Alien constructors). The corresponding alien must be deactivated.

  - add a new explosion to the list of explosions. The radius of the explosion is set to 30. In addition its color must be white if you had a miss, or the "alien color" if you had a hit.
  - If you had a miss, you must decrement the counter by 3 points.

- The game stops when you hit Escape (it will call a function, provided here, that will change the value of a global variable `game_over`), or if the ammunition counter is below or equal to zero. If the game stops, it displays "GAME OVER" around the middle of the canvas.

Class SpaceShip- [10pts]

This class generates a spaceship at the bottom of the screen that you will be able to move using the left or right arrows of your keyboard. Figure 9 shows a snapshot example of what could happen.

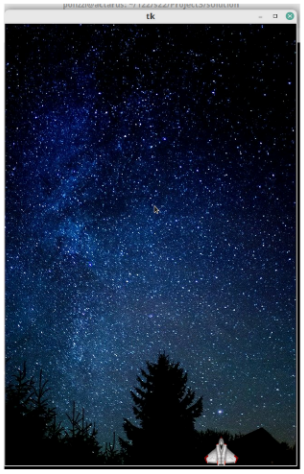

Figure 9: `SpaceShip.py`. Alone in the dark and ready to respond.

How to proceed?

- The main code is already complete.
- Write the entire class `SpaceShip` by implementing: the constructor (only one argument: the canvas), the methods `activate`, `deactivate`, `is_active`, `shift_left` and `shift_right`. Remark: The ship can be found in the file `ship.png`. Also, you will use a private variable `active` (as it was done for the other classes).
- The methods `shift_left` and `shift_right` are "binded" as action to the left and right arrow events in the main code.
- The initial position of the ship (set up using the `activate` method) must be at the bottom middle (be mindful of the height of the ship). the incremental horizontal move of the ship must be 15 pixels.
- When the ship is about to hit (go further) than the left or right boundaries, its move must be limited (less than 15 pixels) so the edge of the ship touches the boundary (and it is not possible to go further to the left or right).

Application game2- [20pts]+[5pts bonus]

Time to implement our second game which regroups features from all the classes! When you start the game, the counter is set to 0 which represents your score. Aliens (red, blue or green) are coming from the top every second. You can move your spaceship by hitting the left and right arrows of your keyboard. If you hit the up arrow, your ship should start shooting missiles from the front of the ship. Once an alien is shot, an explosion of the same color of the alien will take place and your score will increase (with the corresponding alien point value). If you get hit by an alien, your spaceship will explode into a rainbow explosion. There are here two approaches: (i) if you get hit you loose 10 points and you loose the game if your score goes down to zero or below; (ii) **For bonus:** if you get hit, you loose a life (the number of life is represented at the top left by small spaceships, it starts at 3 "life") and you loose the game if you have no life left. Figure 10 shows multiple snapshot examples of what could happen after some time.

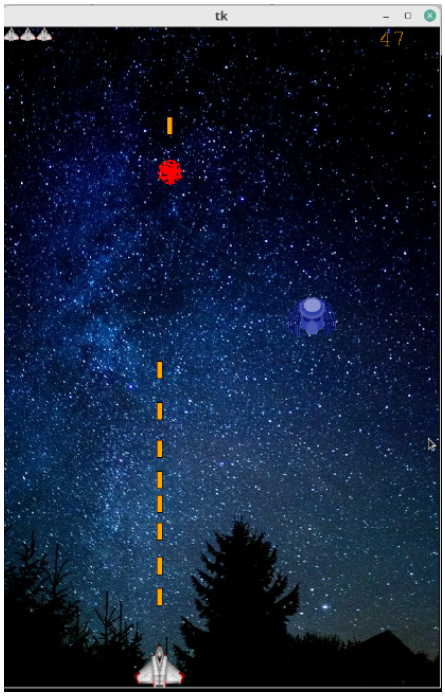

Figure 10: `game2.py`. Space Invaders Revisited!

How to proceed?

- Complete the main method.
- You need to initialize: a list of explosions, a list of missiles, a list of aliens, a counter object which is set at 0 to start with. You also need to instantiate and activate the ship.
- You need to bind the `Left` and `Right` events to `ship_left` and `ship_right` methods. The `Up` event to the `add_missile` static method, and the `Escape` event to the `stop_game` function (like in game1).
- In the while True loop, you need to:
  - generate a new alien every 1s (the time sleep must be set at 0.01s).

- scan the list of explosions and call their `next` method.
- scan the list of missiles and call their `next` method.
- scan the list of aliens and call their `next` method.
- For each alien that you are scanning in the list, you may want to scan the list of missiles to check if this alien is both active and has been shot. If so, you must add an explosion to the list of explosions at the appropriate coordinates and with the appropriate color (size of radius 30), as well as increment the counter with the corresponding alien point value. The alien is also deactivated.
- For each alien that you are scanning in the list, you may also want to test if the ship has been shot. If so, both the ship and the alien are deactivated. The ship must also explode. Here, I use an additional explosion object to keep track of this explosion (separate from the list of explosion). This object is instantiated before entering the while loop (size of radius 50 and rainbow color), and its method `next` is called in the while loop. Hint: In the while loop, you can also check if the status of this explosion object, and the status of the ship, to reactivate the ship if needed. Indeed, in the game after the ship is being shot, it should reappear at the bottom center once its explosion ends (not before).

- For the bonus points: you need to consider adding the small ship life at the top left. In my code, I modified the class `Counter` (modifying the constructor, added some optional arguments and added a method). I also used the method `image.subsample(3, 3)` to scale by 1/3 the width and height of my SpaceShip image.

Complement- Explosion with gravity - [20pts]

We would like to introduce a new more realistic way to implement the explosion taking into account the effect of gravity. Each dots generated will follow a parabolic trajectory over time (dots behave like cannon balls). Each trajectory has its own random angle and random speed.

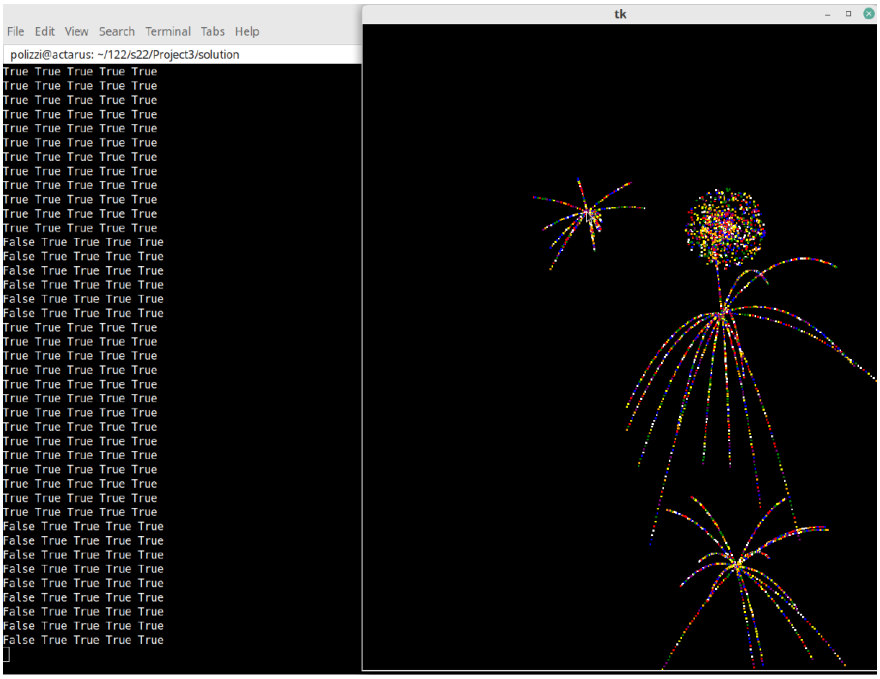

Figure 11: `Explosion.py`. Explosion revisited! Some explosions will become more realistic.

How to proceed?

- In the `Explosion.py` file, create the class `Explosion_gravity` which inherits from `Explosion`.
- This new sub-class contains: the constructor and a method `next` that will be overridden, all other methods are then inherited. It also means that you will need to transform the private variable `active` into a protected variable in the parent class `Explosion` (so it can become accessible by the child class).
- Your new constructor should contain a call to the `super` constructor. At this stage, it makes sense to also generate 15 (maximum dots for a particular time step) random angles from 0 to 359, and 15 random speed from 1 to 5. It is mandatory to use `numpy` to do this, so you can easily generate two numpy arrays `theta` and `speed` using the random numpy method (Hint: the `theta` array can also easily be converted into radians)
- In the `next` method, the x and y positions of the new dots are not random anymore and they must follow the basic Newton gravitation equations:

$$x = x_0 + v\cos(\theta)t$$

$$y = y_0 + v\sin(\theta)t - \frac{g}{2}t^2$$

where x_0, y_0 represent the center of the explosion (defined in the `active` method) of the parent class; t represents the time (you can use the radius variable initialize in the `active`

method to represent the time here), g is our gravity (I use 0.06); θ and v represent the angle and speed of each dot trajectory (defined in your constructor). Hint: in the tkinter window, g is negative in the equation, while v is also negative.

As soon as you enter the next method and if the explosion is active, you can increment your time by 1, and using numpy, you can generate the x and y arrays (containing 15 coordinates) in two lines of code. Once you know the coordinates of these new dots, you can generate them and append them if the list of dots.

- Finally, if the time becomes greater than the max time (you can use the radius number set in the parent constructor), you will call the method deactivate.
- One more thing, you need to update your static method `add_explosion`, and as you did for the Alien class, the method will choose a type of explosions at random between `Explosion` or `Explosion_gravity`. It also means that all your applications `fireworks`, `game1` and `game2` will automatically contains the two types of explosions.
- **Remark:** if you system runs to slow, you can use an increment of 2 (instead of 1) for the time in the method `next`, as well as an increment of 2 (instead of 1) for the radius in the method `next` of the class `Explosion`.

Complement- Some Game Statistics - [30pts]

When playing `game1` and `game2`, it would be interesting to gather some statistics on how many green, blue, red, etc. aliens you have been killing over time. We propose to implement a simple modification of both programs to keep track of those statistics. Let us describe the changes to be made for `game2.py` first. We will proceed in three stages:

1. **Design a dictionary counter**

   - You need to initialize a counter. For this, you will use a dictionary, let us call it `record`, that should use the ID of the aliens (string) as key and the number of kill-shots (int) as values. You should use the color of the aliens as ID (`blue`, `green`, `red`, or `mine` for you own alien bonus point). The values of the dictionary are set to 0 at first.
   - As soon as an alien is shot, you need to increment the corresponding ID in the dictionary. Hint: for your own alien, you may want to use the attribute `color` in your own class Alien to store your own ID.
   - As soon as it is game over, you need to display the dictionary `record` on screen (on the output command prompt, not on the tkinter window), it could look like for example:

```
{'blue': 11, 'red': 14, 'green': 10}
```

   It seems I did better with the red aliens.

2. **Record statistics over time**

   - Basically, we want to create a snapshot of the dictionary values over time. For this, you are going to use a list of tuple (initialize as an empty list at first).
   - Inside the while loop, you are going to append a tuple to this list every 100 time steps (it means every 1s using a time sleep of 0.01s). This tuple of length 3 contains the current values of the red, blue and green indexes of your dictionary (consider a tuple of length 4 if you have your own alien).
   - As soon as it is game over, you are also going to save all of these values into a file named `game2.txt`, use 3 columns separated by a blank space to store these values. A look inside this file may look like:

```
0 0 0
0 0 0
1 0 0
2 0 1
2 0 2
2 0 2
3 0 2
4 0 2
4 1 2
4 1 2
4 1 3
4 1 4
5 1 4
......   (many more lines here)
```

```
12 9 10
12 10 10
12 11 10
12 11 10
13 11 10
14 11 10
14 11 10
14 11 10
```

We will need to remember that the first column is for the red alien shots, then we get the blue and the green (you can add a fourth column if needed). Each row represents 1 second of time.

3. **Display the statistics**

- Write a new file `analysis.py` that will be used to display those statistics using matplotlib and numpy.
- You will call this analysis file using input form the command line as follows (see lecture 3.3):

```
python3 analysis.py game2.txt
```

- Your file is using `game2.txt` as a command line argument, that you will load using `numpy` (lecture 4.2) and plot using `matplotlib` (lecture 4.3). Here is an example of what you should obtain (be mindful of legend, labels, title, etc.)

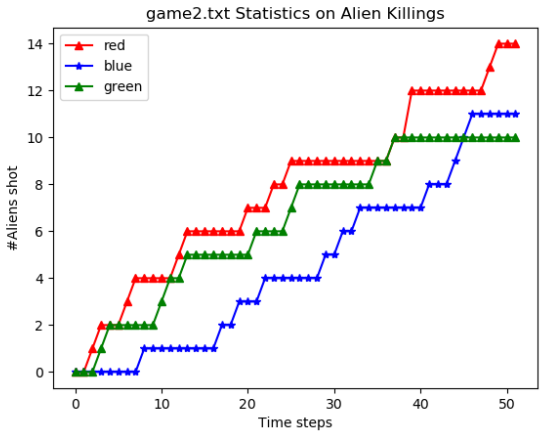

Figure 12: Recorded statistics for my game2 play (around 1 min game, I was pretty slow at killing blue aliens at first)

Finally, you must do exactly the same for `game1.py`. Few differences with game1:

- The `record` dictionary must be passed to the method `shoot` since it will be incremented appropriately in there.

- You will consider the same output dictionary display at game over, and you will use the name `game1.txt` to save your file.
- This is an example of my own final dictionary display:

```
{'blue': 32, 'red': 8, 'green': 9}
```

- You can also run now your statistics from the command prompt:

```
python3 analysis.py game1.txt
```

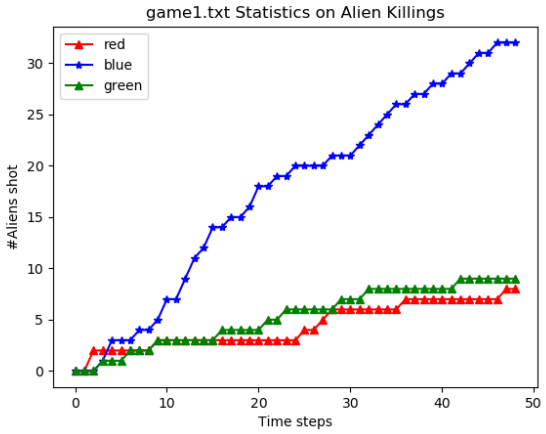

Figure 13: Recorded statistics for my game1 play (it seems I am good at shooting those predators blue faces).