Forum Code Workbook

Problem Set -- Assignment 2

Complete the problems below. You will be graded on the LOs specified in each problem. Some of the problems have optional challenge questions which you may skip. If you complete the optional challenge very well, you ***may*** score an extra ⑤ on the LO specified in the question.

Note: "very well" means answering the whole optional question with no errors or bugs and, it needs to be good enough that it can be published on something like Medium (perhaps with a few minor tweaks). See the CS166 Grading Policy for details.

Problem 1: Design a model

- LOs: #cs166-Modeling
- Suggested word count: 300–600 (please provide the actual word count)

(Adapted from Exercise 2.4 in H. Sayama, *Introduction to the Modeling and Analysis of Complex Systems*, p. 299.)

Question 1 of 11

Write a few paragraphs (no math or code required) describing a real-world system of your choice with many interacting components/agents.

- At which scale do you choose to describe the microscopic components, and what are those components? Briefly describe other scales (larger and/or smaller) at which you could have chosen to model the microscopic state and why your chosen scale is the most appropriate.
- What states can the microscopic components take on and why?
- What are the relationships between the components within the system? How does studying these contribute to the understanding of the overall system?
- How do the states of the components change over time? Some of these changes may depend on interactions between the components, while others may not.

After answering all of the questions above, predict what kind of macroscopic behaviors would arise if you run a computational simulation of your model.

Normal

One real-world system with many interacting components is a human brain. The microscopic components of the brain are neurons, which are specialized cells that communicate with each other using electrical and chemical signals. At this scale, we can study the structure of individual neurons, their electrical properties, and the various types of neurotransmitters that they release.

At a larger scale, we could model the microscopic state of the brain by looking at the activity of individual brain regions or networks of neurons. At a smaller scale, we could study the molecular interactions that occur within individual synapses between neurons. However, modeling the microscopic state at the level of individual neurons is the most appropriate because it captures the basic building blocks of brain function. For example, we would not be able to understand the underlying mechanisms by which the frontal lobe is responsible for movements and language without understanding the relationships between individual neurons in that area at microscopic scale.

The microscopic components of the brain can take on various states, including firing or not firing, releasing different types and amounts of neurotransmitters, and forming new connections with other neurons. These states are determined by a combination of intrinsic properties of the neurons themselves and the patterns of inputs they receive from other neurons.

One specific example is the process of synaptic plasticity, which involves the formation and strengthening of connections between neurons. This process is thought to be a key mechanism underlying learning and memory. During synaptic plasticity, a neuron that receives repeated inputs from another neuron may undergo changes in the strength of its synapses, resulting in the enhanced or decreased ability of the receiving neuron to fire in response to subsequent inputs. This change in the strength of the synaptic connection is thought to be due to a combination of factors, including the release of specific types and amounts of neurotransmitters, the activation of specific receptors on the receiving neuron, and the activation of intracellular signaling pathways.

The relationships between neurons in the brain are complex and can take many forms. Neurons can form excitatory or inhibitory connections with other neurons, which can have a cascading effect on the activity of the network. Other relationships between neurons include excereting different neurotransmitters in the synapse, thus 'communicating' and passing along the signal. Studying these relationships can provide insight into the function of different brain regions and the role they play in various cognitive processes.

The states of the components in the brain change over time due to both intrinsic and extrinsic factors. For example, the firing rate of a neuron may increase in response to a specific sensory input, or it may change over time due to changes in the strength of its synaptic connections with

other neurons. Some of these changes depend on interactions between neurons, while others may be due to changes in the environment or the body. Other states are :

- Plasticity: Neurons in the brain are capable of undergoing changes in their properties over time, a phenomenon known as plasticity. This can involve changes in the strength of synaptic connections, alterations in the expression of different ion channels or receptors, or even the growth of new dendritic spines.
- Metabolic state: The metabolic state of a neuron can influence its activity and function. For example, a neuron that is low on energy may be less likely to fire action potentials or may have reduced plasticity compared to a neuron that is well-nourished.

If we were to run a computational simulation of the brain at the microscopic scale, we would expect to see emergent macroscopic behaviors that are characteristic of brain function. These could include the processing of sensory information, the formation of memories, and the regulation of emotions and behaviors. However, the complexity of the brain means that simulating it accurately is a challenging problem that is still being actively researched.

Words: 621

Problem 2: Predator-prey model

A predator-prey model describes the dynamics of a predator-prey system, describing how the populations of the predator and prey species change over time based on the interactions between the two species

2.1. Creating a Simulation

- LOs: #PythonImplementation, #CodeReadability
- Words: no specific word count; mostly code

Consider a predator-prey system in which a population of predators and preys interact over time. The predator-prey system is simulated on a two-dimensional grid of cells, with each cell representing a patch of habitat. Each cell can be in one of three states: empty, occupied by a predator, or occupied by prey. The population of predators and prey will change over time according to certain rules and parameters.

To start the simulation, you will be using a grid of 35x35 cells, and assume a Moore neighborhood of radius 1 and periodic boundary conditions. The initial population of prey and predators will be randomly placed on the grid, with a 20% probability of the cell being prey and 20% of being the predator.

The state of each cell is updated at each time step according to the following rules:

- The predator will eat neighboring prey and reproduce at the location of the consumed prey with a certain probability of predator birth rate.
- If there are fewer than 2 prey cells in the neighborhood, the predator cell becomes empty.
- The prey will reproduce into empty neighboring cells.
- The predator dies with a certain probability of predator death rate.
- The prey dies with a certain probability of prey death rate.

Note: It is important to apply the rules in the following order:

1. predator eats and reproduces,
2. then prey reproduces,
3. then predator and prey die randomly.

This will ensure that the rules for each type of cell are applied correctly and in the correct order, preventing errors such as a prey reproducing in a cell that has already been occupied by a predator that reproduces earlier.

You will be expected to determine what a reasonable time period designated to run the simulation for.

Your main goal for this question is to create an animation of the simulation showing the 2D grid state and a graph of the population sizes over time. Include your code below. (You will see several code cells which are included in the workbook should you wish to use them.)

Code Cell 1 of 37

```
In [1]  1  # Taken from
        2  #Session 8. (n.d.). Forum. Forum.minerva.edu. Retrieved February 27, 2023, from
           https://forum.minerva.edu/app/courses/2571/sections/10241/classes/72155?course_id=2584
        3
        4  def make_animation(sim, total_frames, steps_per_frame=1, interval=100):
        5      '''
```

Python 3 (1GB RAM) | Edit Run All Cells Kernel Stopped |

```
    -------
    Inputs:
    sim: PredatorPreySimulator() object
        simulation object
    total_frames: int
        the number of time steps shown in the visualization
    steps_per_frame: int
    interval: int
    -------
    Outputs:
    output: HTML() object
        animation of the simulation
    '''
    from matplotlib.animation import FuncAnimation
    from IPython.display import HTML
    from tqdm import tqdm

    def update(frame_number):
        for _ in range(steps_per_frame):
            sim.update() #update the simulation state
        progress_bar.update(1)
        return [sim.observe()] #show the state

    sim.initialize() #initialize the grid
    progress_bar = tqdm(total=total_frames)
    animation = FuncAnimation(
        sim.figure, update, init_func=lambda: [], frames=total_frames, interval=interval)
    output = HTML(animation.to_html5_video())
    sim.figure.clf()
    return output
```

Run Code

Code Cell 2 of 37

In [2]

```
# Adapted from
#Session 8. (n.d.). Forum. Forum.minerva.edu. Retrieved February 27, 2023, from https://forum.minerva.edu/app/courses/2571/sections/10241/classes/72155?course_id=2584

import matplotlib.pyplot as plt
import numpy as np
class PredatorPreySimulator:
    '''
    This class creates the simulation of the predator-prey model by initializing the grid of given dimensions and appropriate distributions of prey cells, predator cells, and empty cells, updating the state of each cell based on the model's rules, and then visualizing the dynamics of the model
    '''
    def __init__(self, p_pred_birth,p_pred_death,p_prey_death, n=35, p_prey=0.2, p_pred=0.2):
        '''
        This method initializes the parameters of the simulation
        -------
        Inputs:
        p_pred_birth: float
            probability of a predator occupying the cell of the prey it just consumed
        p_pred_death: float
            probability of a predator randomly dying and turning into an empty cell
        p_prey_death: float
            probability of a prey randomly dying and turning into an empty cell
        n: int
            the length of one dimension of the cell grid
```

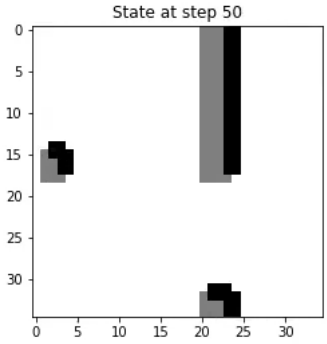
Python 3 (1GB RAM) | Edit Run All Cells Kernel Stopped |

```
        p_prey: float
            initial proportion of prey cells on the grid
        p_pred: float
            initial proportion of predator cells on the grid
        '''
        self.n = n
        self.p_prey = p_prey
        self.p_pred = p_pred
        self.p_pred_birth = p_pred_birth
        self.p_pred_death = p_pred_death
        self.p_prey_death = p_prey_death
        self.step_counter=0
        self.prey_count=[]
        self.pred_count=[]

    def initialize(self,plot=True):
        '''
        This method creates the starting cell grid based on the initial distirbutions of each type of cell
        -------
        Inputs:
        plot: boolean
            a variable to switch the animation off when needed
        '''
        states=[0,1,2] #empty, prey, and predator respectively
        probs=[1-self.p_prey-self.p_pred,self.p_prey,self.p_pred] #initial proportions of empty, prey, and predator cells respectively
        self.config = np.random.choice(states,size=(self.n,self.n),replace=True,p=probs) #create initial grid
        self.prey_count.append(np.count_nonzero(self.config == 1))
        self.pred_count.append(np.count_nonzero(self.config == 2))
        if plot==True:
            self.figure, self.axes = plt.subplots()



    def observe(self):
        '''
        This method creates the video visualization of the predator prey model dynamics on the cell grid
        '''
        plot = self.axes.imshow(
            self.config, vmin=0, vmax=2, cmap = 'binary')
        self.axes.set_title(f'State at step {self.step_counter}')

        return plot

    def update(self):
        '''
        This method updates each of the cells' state based on the provided model rules by checking the current state of the cell and the states of its neighbours (Moore neighbourhood with radius 1)
        '''

        self.nextconfig=np.zeros([self.n,self.n]) #initialize the next state grid

        #for the rule 1: predator eats and reproduces
        for x in range(self.n):
            for y in range(self.n):
                state = self.config[x,y] #take the state of the current cell

                if state==2: #if the current cell is predator
                    neighbours=[]
                    #check each neighbour
                    for dx in [-1, 0, 1]:
                        for dy in [-1, 0, 1]:
```

```
83
84                          #if the neighbour is prey, consume it and populate its cell with a given probability
85                          if self.config[(x + dx) % self.n, (y + dy) % self.n]==1:
86                              self.nextconfig[(x + dx) % self.n, (y + dy) % self.n]=2 if np.random.random() <
self.p_pred_birth else 0
87
88                      #predator dies if there is less than 2 prey in its neighbourhood
89                      if neighbours.count(1)<=1:
90                          state=0
91
92                      self.nextconfig[x,y] = state #update the state of the current cell accordingly
93
94
95          #for the rule 2: then prey reproduces
96          for x in range(self.n):
97              for y in range(self.n):
98                  state = self.config[x,y] #take the state of the current cell
99
100                 if state==1 and self.nextconfig[x,y]!=2: #if the current cell was prey and remained prey (wasn't eaten)
101                     self.nextconfig[x,y] = state #keep this cell as a prey
102
103                     #check all neighbours
104                     for dx in [-1, 0, 1]:
105                         for dy in [-1, 0, 1]:
106                             #if there is an empty cell around, populate it
107                             if self.nextconfig[(x + dx) % self.n, (y + dy) % self.n]==0:
108                                 self.nextconfig[(x + dx) % self.n, (y + dy) % self.n]=1
109
110
111         #for the rule 3:then predator and prey die randomly
112         for x in range(self.n):
113             for y in range(self.n):
114
115                 #a predator randomly dies with a certain probability
116                 if self.nextconfig[x,y]==2:
117                     self.nextconfig[x,y]=0 if np.random.random() < self.p_pred_death else 2
118                 #a prey randomly dies with a certain probability
119                 if self.nextconfig[x,y]==1:
120                     self.nextconfig[x,y]=0 if np.random.random() < self.p_prey_death else 1
121
122
123         self.prey_count.append(np.count_nonzero(self.nextconfig == 1)) #count the prey
124         self.pred_count.append(np.count_nonzero(self.nextconfig == 2))#count the predators
125         self.config = self.nextconfig #update the next configuration to be the current configuration for the next step
126         self.step_counter += 1 #mark one time step
127
128
129 sim = PredatorPreySimulator(1,0,0) #create an instance of a simulation
130 make_animation(sim, total_frames=50, steps_per_frame=1)#show the visualization
```

Run Code

```
Out [2]     100%|■■■■■■■■■■| 50/50 [00:06<00:00,  7.93it/s]

            <Figure size 432x288 with 0 Axes>
```



Code Cell 3 of 37

In [3]
```
def run_animation(p_pred_birth,p_pred_death,p_prey_death,n=35,total_frames=50):
    '''
    This function is just to run and visualize the simulation
    -------
    Inputs:
    p_pred_birth: float
        probability of a predator occupying the cell of the prey it just consumed
    p_pred_death: float
        probability of a predator randomly dying and turning into an empty cell
    p_prey_death: float
        probability of a prey randomly dying and turning into an empty cell
    n: int
        the length of one dimension of the cell grid
    total_frames: int
        the number of time steps shown in the visualization
    -------
    Outputs:
        animation
    '''
    sim = PredatorPreySimulator(p_pred_birth,p_pred_death,p_prey_death,n=n) #create an instance of a simulation
    return make_animation(sim, total_frames, steps_per_frame=1)#show the visualization sim.initialize
```

Run Code

Code Cell 4 of 37

In [4]
```
def run_simulation(p_pred_birth,p_pred_death,p_prey_death,n=35,total_steps=50):
    '''
    This function is just to run the simulation
    -------
    Inputs:
    p_pred_birth: float
        probability of a predator occupying the cell of the prey it just consumed
    p_pred_death: float
        probability of a predator randomly dying and turning into an empty cell
    p_prey_death: float
        probability of a prey randomly dying and turning into an empty cell
    n: int
```

```
        the length of one dimension of the cell grid
    total_frames: int
        the number of time steps shown in the visualization
    -------
    Outputs:
    sim: PredatorPreySimulator() object
        the simulation object
    '''
    sim = PredatorPreySimulator(p_pred_birth,p_pred_death,p_prey_death,n=n) #create an instance of a simulation
    sim.initialize(plot=False) #initialize
    for i in range(total_steps):
        sim.update() #update
    return sim
```

Run Code

Code Cell 5 of 37

```
In [5]  1  # your code here
```

Run Code

2.2. Analyzing the simulation

- LOs: #EmpiricalAnalysis, #TheoreticalAnalysis, #cs166-Modeling

Select three distinct sets of parameter values for the predator birth rate, predator death rate, and prey death rate of your choice so that you can study interesting regimes in this system. Run the simulation multiple times for each set of parameters, and then answer the following questions.

Question 2 of 11

(a) Identify at least two edge cases where you can argue that the simulation is implemented as intended.

Normal

1) One simple edge case would be to set the parameters predator birth rate, predator death rate, and prey death rate to be 0, 1, and 1 respectively. It would mean that there is no chance for the predator to occupy the consumed prey's cell, and there is a 100% chance of both predator and prey cells dying. In other words, it means that all predator and prey cells should die immediately after the first time step. It makes sense intuitively because if there is 100% probability of predators and prey dying, plus there is no way for the predator to occupy pre's cell, then everyone will die instantaneously. This is exactly what we see in the visualization : starting with time step 1, all cells are white, meaning all predators and preys died immediately as we started the simulation. The grid remains all white (all cells empty) until the end of the simulation. This supports the validity of our simulation.

2) Another edge case would be to set the parameters predator birth rate, predator death rate, and prey death rate to be 0, 1, and 0 respectively. It would mean that there is no chance for the predator to reproduce in the consumed prey cell, predator would die with 100% probability, and prey will never randomly die (can only be eaten by predators). What we see in the simulation with these parameters is that after the very first time step, the simulation reaches the state with almost all prey (grey cells) and a small proportion of empty cells (white cells). When we transition from the initial state to the the very first time step, we expect predators to eat nearby prey (turning those cell into empty as predators can not reproduce), prey to populate the nearby empty cells and then all predators to die due to 100% death rate as a parameter. The reason the number of prey would immediately grow after the first time step is because they also can not die randomly, and they will populate every nearby empty cells. Although they can be consumed by the predators, in which case their cell becomes empty, they will likely become repopulated by a neighbouring prey since predators can not reproduce. Now when we go from the first time step to the second, all predators died and left empty cell instead of them, which in this second time step are populated by nearby prey. Thus, in a few time steps, the prey would populate the whole grid. Since there are no more predators, and prey can not die randomly and there is nowhere else to populate, the state remains constant for the rest of the simulation. We can see that the simulation results are the same as we would intuitively expect

3)The last edge case would be to set the parameters predator birth rate, predator death rate, and prey death rate to be 1, 0.1, and 0.9 respectively. It means that the predators would populate the cells of the consumed prey all the time, predators have a very low probability of random death, and prey has a very high probability of random death. What we would expect to happen given the parameters is the prey population to decrease dramatically after the first time step due to high random death rate and consumption by predators, while the predator population should increase somewhat given that they always reproduce into consumed prey cells and the probability of death is very low. After the second time step and for the remaining time steps we would expect both the predator and prey populations to decline rapidly due to the high prey random death chance and hence, lack of prey for the predators. Thus would lead to a quick die out of both populations. This is exactly what we observe in the simulation visualization below: at the first time step, the population of predators in increased, and for the following time steps both populations are decreasing until none left.

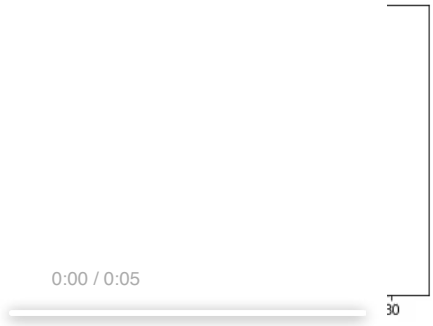
Thus, we can see that for the 3 edge cases chosen, the simulation results coincide with the expected result, suggesting the simulation is implemented correctly .

Code Cell 6 of 37

In [6]:
```
#edge case 1
run_animation(0,1,1)
```

Run Code

Out [6]:
```
100%|██████████| 50/50 [00:06<00:00,  8.14it/s]
<Figure size 432x288 with 0 Axes>
```

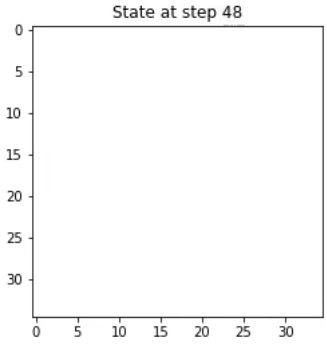


Code Cell 7 of 37

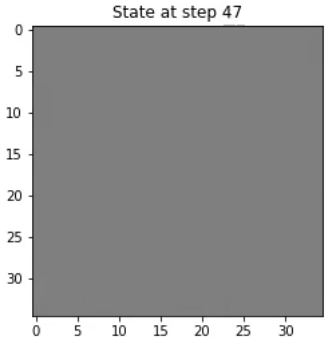
In [7]:
```
#the first displayed animation, edge case 3
display(run_animation(1,0.1,0.9))
#the second displayed animation, edge case 2
display(run_animation(0,1,0))

```

Run Code

Out [7]:
```
100%|██████████| 50/50 [00:06<00:00,  8.13it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
```




```
100%|██████████| 50/50 [00:06<00:00,  7.83it/s]
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```



Question 3 of 11

(b) Create a graph of the population size over time, including the 95% confidence interval of the mean for each of the three sets of parameters you have identified. Analyze and interpret the empirical results by comparing the population dynamics and the behavior of the system under different parameter values.

Normal

The 3 sets of parameters predator birth rate, predator death rate, and prey death rate are respectively [0.6,0.8,0.8], [0.2,0.8,0.8], and [0.2, 0.2, 0.2].

1) For the first set of parameters - [0.6,0.8,0.8], - we can see from the graph below that the predator population quickly die out, while prey populations decreases to around 120, but then grows to around 160 and stabilizes. We can explain the death of the predators and low count of prey by the high random death rate. We can also explain the fact that the prey population does survive by considering the fact that they can reproduce in neighbouring empty cells, which is the majority of cell when we initialize the simulation. The reason for why prey population remains relatively stable is because after a few time steps, only prey is remaining (no predators), which means that the only actions that happen is all prey reproduces in all neighboring empty cells, and then prey reduces with the probability of 0.8. Thus, the grows and death rates are approximately balanced, leading to a somewhat stable state.

Considering the 95% confidence interval (CI) of the mean for this case, we can barely see it for the prey population and can not see it for the predators. The reason is that it is very narrow, meaning we are absolutely certain in the mean count of 0 for predator population, and almost absolutely certain the the mean count of prey population. This happens because no matter how many time we rerun the simulation, we always get the mean of predator population to be 0 just after a few time steps, which stays 0 till the end of the simulation.

For the prey population, we are very confident in its mean due to a large number of simulations. The more times we run the simulation (100 times), the smaller the CI will be because the standard error of the mean decreases as the number of simulations increases, which we can see from the standard error definition. The standard error of the mean is proportional to the standard deviation of the population divided by the square root of the sample size (number of simulations). As we increase the sample size by running more simulations, the standard error of the mean decreases, which means that the confidence intervals become narrower. This is because the larger sample size provides more information about the population distribution, which reduces the uncertainty in our estimates.

2) For the first set of parameters - [0.2,0.8,0.8] - we can see from the graph below that the overall dynamics is the same as for the case with [0.2,0.8,0.8] parameters, with the only difference that the prey population does not dip to around 120 and then grows to 160, but rather decreases to 160 straight away and remains stable there. This difference can be explained by the change of the predator birth parameter. In the previous example, the birth parameter was higher, so more eaten prey was occupied by predators, meaning prey can not reproduce in that cell anymore. In this case, however, the birth rate is quite low, meaning more eaten prey cells remain empty, which allows prey to populate those cells. Thus, since the prey can potentially populate more steps in the lower birth rate case, it does not decrease as much as when the birth rate is higher. Also note that these patterns are only relevant in the first few time steps because that's when the predators still exist (and this can eat prey and influence the overall dynamics). After the predators die, they can not eat prey and the birth rate parameter is irrelevant. The only relevant parameters left are death rates of both populations, which are the same in this and previous case. This explains why in this and previous case the prey population stabilizes and approximately the same count (around 160).

Considering the 95% confidence interval (CI) of the mean for this case, we observe the same pattern as in the previous case and explain it the same way.

3) For the last set of parameters - [0.2, 0.2, 0.2] - we can see that the prey population starts to increase up to 800 and the predator population - decrease down to around 150 in the first few time steps. This is explained by the fact that prey starts to populate empty cells (which are initialized ot be the majority of the grid) and the predators die due to the lack of prey. Both populations have an equal and small random death rate, so it does not cause major differences between the populations. Aftet the prey reaches around 800, the predators have a lot of prey around, so they stop dying as much from the lack of prey and instead consume it and reproduce, although with the small rate. The increase in prey count explains the increase in predator population that follows it. The increase is not too large (from around 150 to 350) since the birth rate is only 0.2. Consequently, since the predator population increases due to the consumption of prey, the prey population decreases a little bit to around 600. It is at this point that the system reaches somewhat stable state, which is seen as flat lines on the graph for later time steps.

Comparing this case to the previous case, [0.2,0.8,0.8], where the changes are in the death rates while birth rate remained the same, we see that the overall count of both population is much bigger : the y scale of the previous case graph is 0-250, while here is 100-800. This is explained by a much smaller death rate, allowing both populations to reproduce more. Another change is that in this case, the predator populations survives rather than dies out, which is again explained by a smaller random death rate and higher availability of prey.

Comparing this case to the first case, [0.6,0.8,0.8], we can see an overall increase in both populations count due to the lower death rates. We can also see that unlike in the first case, there is no initial decrease in prey population. This is because in the first case, the prey population initially decreased due to a higher birth rate (0.6), while in this case the birth rate is quite low (0.2), resulting in the prey growth rate due to reproduction into empty cells being higher than the death rate by predator consumption.

Considering the 95% confidence interval (CI) of the mean for this case, we observe the same pattern as in the previous case and explain it the same way. Here we can not even see it at all as we are even more certain in the mean prey and predator counts due to the large number of simulations made and the specifics of the parameters that influence a stable state.

Code Cell 8 of 37

In [8]

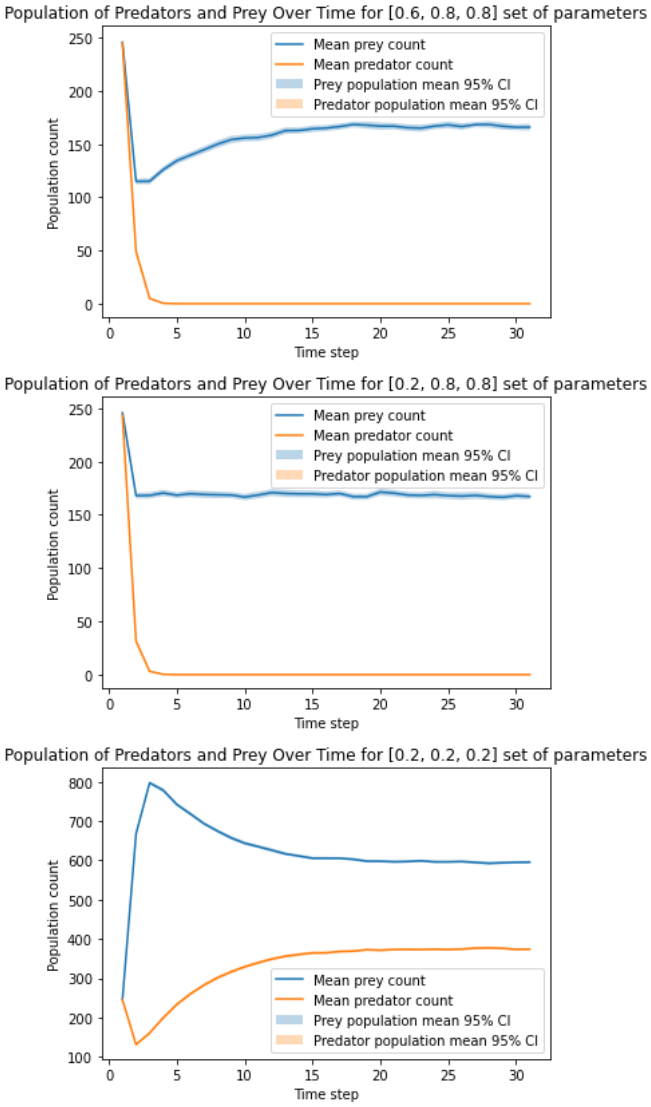
```
import scipy.stats as sts
import seaborn as sns


def pop_size_over_time(p, total_steps):
    '''
    This functon takes a set of parameters (predator birth rate and death rates of both populations) and the number of simulation steps as inpputs, runs the simulation 100 times, and records the results for each simulation  - list of the progression of the prey and predator population sizes over the course of the simulation
    ------
    Inputs:
    p: list
        a set of parameters for the predator birth rate and death rates of both populations
    total_steps: int
        a number of steps the simulation should run for
    ------
    Outputs:
    [prey_count_list, pred_count_list] : list
        a nested list of prey population sizes (for each of the 100 simulations) list and predator population sizes (for each of the 100 simulations) list
    '''
    #lists to store the results of each simulation
    prey_count_list=[]
```

Python 3 (1GB RAM) | Edit Run All Cells Kernel Stopped |

```
    for i in range(100): #simulate 100 times for the same set of parameters
        sim = run_simulation(p[0], p[1], p[2],total_steps=total_steps)   #simulate
        prey_count_list.append(sim.prey_count) #store prey count list
        pred_count_list.append(sim.pred_count) #store predator count list

    return [prey_count_list, pred_count_list]


def plot_conf_int(p, total_steps):
    '''
    This functon takes a set of parameters (predator birth rate and death rates of both populations) and the number of simulation steps as inpputs to pass to the pop_size_over-time function, averages both population sizes at each time step over the 100 simulation, and plots these mean population sizes alogside their corresponding 95% CIs
    ------
    Inputs:
    p: list
        a set of parameters for the predator birth rate and death rates of both populations
    total_steps: int
        a number of steps the simulation should run for
    ------
    Outputs:
    graphs of the mean prey and predator population size progression with corresponding 95% CI for a given set of parameter values
    '''
    counts=pop_size_over_time(p,total_steps) #assign the nested list of population sizes to a variable

    av_prey_count=np.mean(counts[0], axis=0) #find the mean prey population size over time
    av_pred_count=np.mean(counts[1], axis=0)#find the mean predator population size over time

    t_prey = sts.sem(counts[0], axis=0)#find the standard error of the prey population sizes
    t_pred = sts.sem(counts[1], axis=0)#find the standard error of the predator population sizes

    x=np.arange(1, total_steps+2) #x axis

    plt.plot(x, av_prey_count,label='Mean prey count') #plot mean prey population size
    plt.fill_between(x, av_prey_count + 1.96*t_prey, av_prey_count- 1.96*t_prey, alpha=.3,label = 'Prey population mean 95% CI')#95 CI

    plt.plot(x, av_pred_count,label='Mean predator count') #plot mean prey population size
    plt.fill_between(x, av_pred_count + 1.96*t_pred, av_pred_count- 1.96*t_pred, alpha=.3,label ='Predator population mean 95% CI')#95CI

    plt.title(f'Population of Predators and Prey Over Time for {p} set of parameters')
    plt.xlabel('Time step')
    plt.ylabel('Population count')
    plt.legend()

    plt.show()


plot_conf_int([0.6,0.8,0.8], 30)
plot_conf_int([0.2,0.8,0.8], 30)
plot_conf_int([0.2,0.2,0.2], 30)
```

Run Code

Out [8]



Code Cell 9 of 37

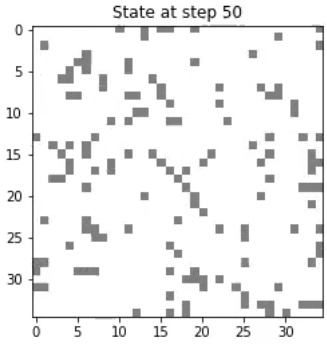
In [9]

```
1 #show animations for the cases above
2 display(run_animation(0.6,0.8,0.8))
3 display(run_animation(0.2,0.8,0.8))
4 display(run_animation(0.2,0.2,0.2))
5
```

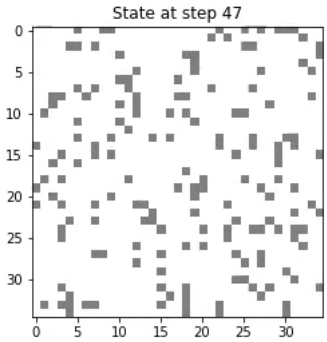
Run Code

Out [9]

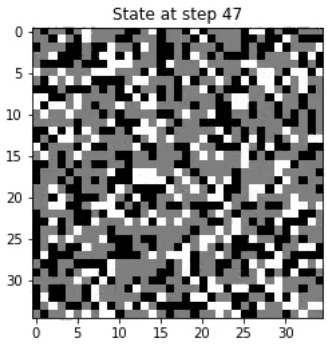
```
100%|██████████| 50/50 [00:06<00:00,  7.65it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
```




```
100%|██████████| 50/50 [00:06<00:00,  7.82it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
```




```
100%|██████████| 50/50 [00:07<00:00,  7.09it/s]
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```



Code Cell 10 of 37

```
In [9]   1
```

Run Code

Question 4 of 11

(c) Run your simulation for vanishing prey death rate, predator death rate, and predator birth rate, and produce a new plot to illustrate the time evolution of the prey and predator population in this instance. Interpret the results you obtained.

Normal

This edge case implies setting the parameters predator birth rate, predator death rate, and prey death rate to be 0, 0, and 0 respectively. It would mean that there is no chance for the predator to reproduce in the consumed prey cell, as well as both prey and predator would never randomly die. What we see in the simulation with these parameters is that after the first few time steps, the simulation reaches a stable state with all prey (grey cells) and a small proportion of predators (black cells). When we transition from the initial state to the the very first time step, we expect all prey to populate the neighboring empty cells and predators to remain close to the initial amount, with some predators dying due to the lack of prey in the neighbourhood. The reason that predators concentration should remain approximately the same (not just after the first time step, but forever) is because predators can not die randomly, can not die due to prey lack because after the first few time steps (since the prey would populate all possible empty cells), and can not reproduce into prey cells. The reason the number of prey would grow to its maximum very quickly after the first few time steps is because they also can not die randomly, and they will populate every empty cell in just a couple of steps. Although they can be consumed by the predators, in which case their cell becomes empty, they will immediately become repopulated by a neighbouring prey since predators can not reproduce.

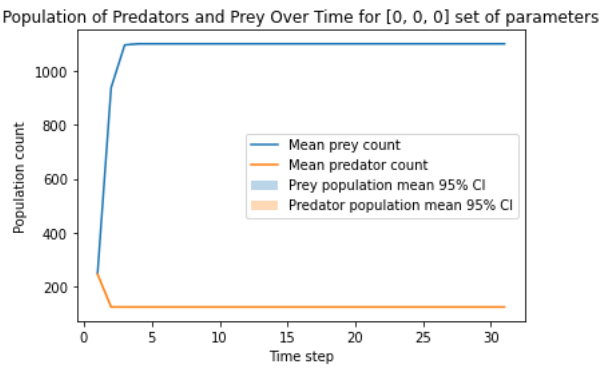
Thus, we expected that the grid will be populated by prey very quickly, with occasional predator cells remaining, and we also expected this grid state to remain stable, given that neither prey nor predators can die and predators can not reproduce. This is exactly what we observed in a simulation, again supporting its validity. The plot also shows how the prey populations rapidly grows and predator population rapidly decreases, after which both remain perfectly stable. We are also extremely confident in this result since 95% CI is extremely narrow that we can not even see it. This is explained using the same reasoning as before - as we increase the sample size by running more simulations, the standard error of the mean decreases, which means that the confidence intervals become narrower.

Code Cell 11 of 37

In [10]:
```
#plot the vanishing parameters edge case
plot_conf_int([0,0,0], 30)
```

Run Code

Out [10]

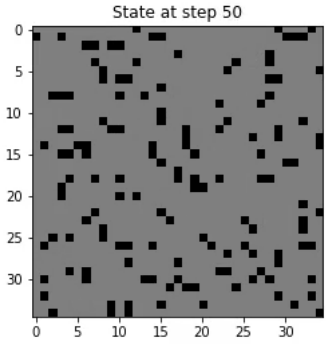


Code Cell 12 of 37

In [11]:
```
#visualize the vanishing parameters edge case
run_animation(0,0,0)
```

Run Code

Out [11]
```
100%|██████████| 50/50 [00:06<00:00,  7.73it/s]

<Figure size 432x288 with 0 Axes>
```



Code Cell 13 of 37

```
In [12]  1  # your code here
```

Run Code

Question 5 of 11

(d) Explicitly compare your simulation results with the results obtained from this simulation.

Normal

Parameters [0, 1, 1]:
Just like in our simulation, the results obtained from the online simulation show that both prey and predators die immediately after the first time step.

Parameters [0, 1, 0]:
Just like in our simulation, the results obtained from the online simulation show that prey population quickly fills the whole grid in just a few time steps.

Parameters [1, 0.1, 0.9]:
Just like in our simulation, the results obtained from the online simulation show that both prey and predators die out in a few time steps. Same as in our simulation, online version shows that after the first time step, predator population grows and prey population largely declines, after which predator population also starts to decline together with prey population due to the lack of prey availability.

Parameters [0.6,0.8,0.8]:
Just like in our simulation, the results obtained from the online simulation show that both prey and predators populations start to dramatically decrease, after which predator population remains at 0, while prey population grows back and stabilizes at around 150.

Parameters [0.2,0.8,0.8]:
Just like in our simulation, the results obtained from the online simulation show that predators population start to dramatically decrease and prey population decreases too just not as dramatic, after which predator population remains at 0, while prey population grows back and stabilizes at around 150.

Parameters [0.2, 0.2, 0.2]:
Just like in our simulation, the results obtained from the online simulation show that predators population start to decrease and prey population increase, after which predator population increases and stabilizes at around 150-350, while prey population grows back and stabilizes at around 650.

Parameters [0, 0, 0]:
Just like in our simulation, the results obtained from the online simulation show that predators population start to decrease slightly and prey population increases dramatically, after which both populations remain constant

The slight differences in prey and predator counts between our and online simulation can be explained by the difference in the initial concentration of prey and predators.

Overall, we see that the results from our simulation match the results from the online simulation, providing additional evidence that our simulation is implemented correctly.

Code Cell 14 of 37

```
In [13]  1  # your code here
```

Run Code

Code Cell 15 of 37

```
In [14]  1  # your code here
```

Run Code

Code Cell 16 of 37

```
In [15]  1  # your code here
```

Run Code

Question 6 of 11

(e) The Lotka-Volterra equations are a set of mathematical equations that describe the dynamics of a predator-prey system. These equations can be used to predict the population growth and decline of both the predator and prey populations over time.

$$\frac{\mathrm{d}p(t)}{\mathrm{d}t} = \alpha\ p(t) - \beta\ p(t)\ P(t) \qquad \text{Eq. (1)}$$

$$\frac{\mathrm{d}P(t)}{\mathrm{d}t} = \delta\ p(t)\ P(t) - \gamma\ P(t) \qquad \text{Eq. (2)}$$

where p is the prey population, P is the predator population, and $\{\alpha,\beta,\gamma,\delta\}$ is a set of (positive) interaction parameters, all constant in time.
Offer an intuitive explanation for why these equations are a sensible parametrization of the predator-prey system. Explain what are the variables, update rules, and any underlying assumptions important for this analytical model to be a good starting point for describing this system.

Normal

The Lotka-Volterra equations are a sensible parametrization of the predator-prey system because they capture the essential dynamics of this ecological interaction. The model assumes that the prey population grows exponentially in the absence of predators, but is subject to predation by the predator population. The predator population, in turn, depends on the availability of prey for sustenance. This results in a cyclic pattern of population dynamics, with the predator population increasing as the prey population grows, but declining when the prey population is depleted. This feedback loop creates oscillations in the population sizes of both species. In other words, the rate of prey's population change is defined as the prey's population growth rate minus the consumption by the predators rate, while the rate of predators' population change is defined as the predators' population growth rate (depending on prey consumption) minus the predators' natural death rate. We can see that these equations are a sensible representation of reality as they update the populations' sizes based on relevant parameters and integrate necessary interdependencies between the populations, all of which is happening in reality, although highly simplified in the equations.

The variables in the Lotka-Volterra model are the prey population (p) and the predator population (P). In the equations, P(t) and p(t) represent the corresponding population sizes at a certain time step. Apart from these variables, the equations contain a set of constant parameters, which are specific to a given model.

The updating rules for the model are given by the differential equations above, where α is associated with the growth rate of the prey population, β - the predation rate, δ - predator population growth (based on consuming the prey, but not perfectly correlated), and γ - the random death rate of the predator population. We can see that these updating rules integrate the essential interactions and dependencies between the populations, such as predators' population needs prey presence to grow.

The underlying assumptions of the model are important to consider when using it to describe real-world predator-prey systems. One key assumption is that the populations are well-mixed and homogeneous, so that there is no spatial structure or variation in the environment. This is a simplification that may not be realistic in many cases, as the quality and availability of resources may vary across space. Another important assumption is that the population sizes are continuous and can be described by differential equations. In reality, populations are discrete and subject to stochastic events such as births, deaths, and immigration/emigration. Additionally, the model assumes that the predator and prey populations interact only with each other, neglecting other factors that may affect their dynamics such as disease, competition, and mutualism. Despite these simplifications, the Lotka-Volterra model provides a useful starting point for understanding the basic mechanisms of predator-prey interactions and has been applied successfully to many ecological systems. In addition, the Lotka-Volterra model incorporates several other assumptions that may not reflect the natural world accurately, particularly concerning the environment and the development of predator and prey populations. These include the following:

- The prey population always has sufficient food available.
- The food supply of the predator population is solely dependent on the number of prey available.
- During the modeling process, the environment does not favor one species over another, and genetic adaptation is irrelevant.
- Predators can consume prey infinitely

Question 7 of 11

(f) Use a numerical method such as Euler's to solve the Lotka-Volterra equations and study the system's behavior. For this, choose appropriate values for the parameters α,β, γ, δ in the Lotka-Volterra equations, as well as initial population values for the predator and prey populations. You should include that information in the text box provided below. You can use a range of

a. 0.5 to 0.8 for α,
b. 0.0001 to 0.001 for β,
c. 0.2 to 0.5 for γ and
d. 0.0001 to 0.001 for δ as a starting point, but keep in mind that the actual parameter values may fall outside of these ranges depending on the specific system being modeled.

Code cells are provided for you to apply the numerical methods to solve the system's dynamics.

Normal

For example, we can see a plot below for the parameters values : alpha = 0.7, beta = 0.0005, delta = 0.0001, gamma = 0.4. The initial populations are set to be the same as in the simulations before (245 prey and predators). The resulting population dynamics we see on the graph is periodic, with both prey and predator populations oscilating from higher to lower. We can see that when prey is growing smoothly (up to around 26000), the predators stay at a relatively low count (close to 0). Then at certain point the prey population rapidly and dramatically falls to almost 0 (1, to be exact) (as a result of predation). As the prey count falls, the predator population rapidly grows, but not too much (from around 0 to 6000). Lastly, the prey count stays that low for some time, and predator count slowly goes down due to the lack pf prey availability until the predator and prey count reach the initial counts, at which point the cycle repeats again.

For different parameter values, the cyclical nature still persists. The only difference is in the relative peaks of prey and predator populations, as well as the rates at which they increase/decrease.

One observation that might not hold in reality is that when the prey population reaches almost 0, it eventually recovers. In real world, if the population reaches 1 organism, it is almost impossible to restore the initial population size and the prey dies out. As the prey dies out, the predators also die out since they do not have anything else to eat (assuming this prey is their main and only resource). Thus, in real world we would expect the prey to never restore its population , and predators would also likely decrease is size dramatically due to the lack of prey. We do not observe it in the model.

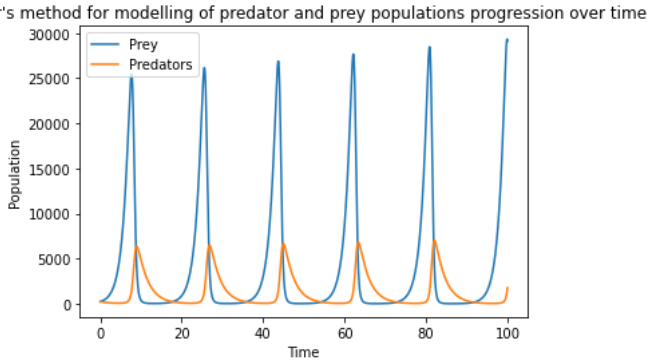
Code Cell 17 of 37

```
In [16]  1 import numpy as np
         2 import matplotlib.pyplot as plt
```

```
def predprey_num_sol(alpha = 0.7,beta = 0.0005,delta = 0.0001, gamma = 0.4,Prey_0 = 245,Pred_0 = 245,dt = 0.01,num_steps = 10000):
    '''
    This function executes Lotka-Volterra equations with given parameter values, initial states, and the number of steps. Then it plots the resulting dynamics of the prey and predator populations.
    -------
    Inputs:
    alpha: float
        prey growth rate parameter
    beta: float
        predation rate parameter
    gamma: float
        predator natural death rate parameter
    delta: float
        predator growth rate parameter
    Prey_0: int
        starting prey population
    Pred_0 : int
        starting predator population
    dt: float
        time step
    num_steps: int
        number of steps
    -------
    Outputs:
    a graph showing the populations' progression over time
    '''

    # Initialize arrays to store population values
    Prey = np.zeros(num_steps)
    Pred = np.zeros(num_steps)

    # Set initial values
    Prey[0] = Prey_0
    Pred[0] = Pred_0

    # Apply Euler's method for a set number of steps
    for i in range(1, num_steps):
        dPrey_dt = alpha*Prey[i-1] - beta*Prey[i-1]*Pred[i-1]
        Prey[i] = Prey[i-1] + dt*dPrey_dt

        dPred_dt = delta*Prey[i-1]*Pred[i-1] - gamma*Pred[i-1]
        Pred[i] = Pred[i-1] + dt*dPred_dt

    # Plot population evolution
    t = np.linspace(0, num_steps*dt, num_steps)
    plt.plot(t, Prey, label='Prey')
    plt.plot(t, Pred, label='Predators')
    plt.xlabel('Time')
    plt.ylabel('Population')
    plt.legend()
    plt.title("Euler's method for modelling of predator and prey populations progression over time")
    plt.show()

predprey_num_sol()
```

Run Code

Out [16]



Code Cell 18 of 37

```
In [17]  1  # your code here
```

Run Code

Code Cell 19 of 37

```
In [18]  1  # your code here
```

Run Code

Code Cell 20 of 37

```
In [19]  1  # your code here
```

Run Code

Code Cell 21 of 37

```
In [20]  1  # your code here
```

Run Code

Question 8 of 11

(g) [Optional Challenge] Interpret the empirical results you obtained in light of this theoretical modeling. Make sure to make a thorough comparison between the expected theoretical results and the simulation results you obtained. Are there specific regimes where the theoretical approximation is particularly good, or bad? Explain your reasoning.

Normal

Code Cell 22 of 37

```
In [21]  1 # your code here
```

Run Code

Code Cell 23 of 37

```
In [22]  1 # your code here
```

Run Code

Code Cell 24 of 37

```
In [23]  1 # your code here
```

Run Code

Code Cell 25 of 37

```
In [24]  1 # your code here
```

Run Code

Code Cell 26 of 37

```
In [25]  1 # your code here
```

Run Code

Problem 3: Simulating Dice Rolls

- LOs: #EmpiricalAnalysis, #PythonImplementation, and #CodeReadability
- Optional LO: #TheoreticalAnalysis

(Adapted from Problem 3 Shonkwiler p.43)

(a) Simulate 1,000 rolls and 10,000 rolls (separately) of a pair of dice in Python 🎲🎲 and, for each roll, record the following information:

- The sum of the two dice,
- The individual pair, e.g. (1, 1), (1, 2), ..., (6, 6). Please note that the individual pair does distinguish between (a, b) and (b, a)--they are **not** regarded as the same.

Code Cell 27 of 37

In [26]:

```
import numpy as np
import random

def simulate_dice_rolls(n_dice,n_rolls):
    """
    Simulate rolling a set number of dice for a set number of times. The function stores the sum of each set and the
    individual dice values for each roll in two separate lists.
    -------
    Inputs:
    n_dice: int
        the number of die rolls in a trial
    n_rolls: int
        the number of times we want to roll certain number of dice
    -------
    Outputs:
    rolls: list
        a list of integers representing the sum of each set of dice rolls for a set umber of rolls.
    pairs: list
        a list of lists representing the individual dice values for each roll for a set number of rolls.
    """
    # store sums and sets
    rolls = []
    sets = []
     # Simulate n_rolls rolls
    for i in range(n_rolls):
        sum_rolls=0
        values=[]
        #rolls a die a given number of times
        for j in range(n_dice):
           #Generate random integer between 1 and 6 (inclusive) to simulate rolling a die.
            dice = random.randint(1, 6)
            sum_rolls+=dice
            values.append(dice)

        rolls.append(sum_rolls) # Add the sum of the dice
        sets.append(tuple(values)) # Add the individual dice values.

    return [rolls, sets]
```

Run Code

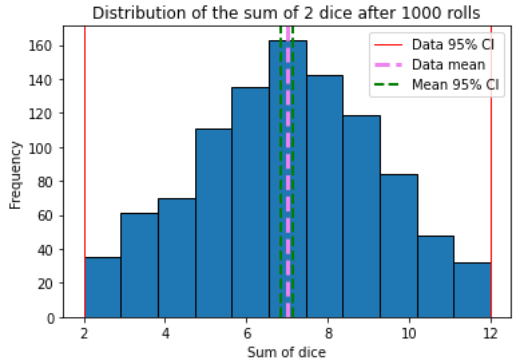
Code Cell 28 of 37

In [27]:
```
from scipy.stats import t
def print_hist_dice(n_dice,n_trials,n_bins=11):
    '''
    The function plots the histogram of the dice roll sums after a given number of trials for a given number of dice
    ------
    Inputs:
    n_dice: int
        the number of dice to be rolles in one go
    n_trials: int
        the number of rolls performed
    n_bins: int
        the number of bins in a histogram
    ------
    Outputs:
        a histogram of the the dice roll sums after a given number of trials for a given number of dice with relevants CIs
    '''
    # Calculate mean and confidence intervals
    rolls=simulate_dice_rolls(n_dice,n_trials)[0]
    mean= np.mean(rolls)
    std_err= np.std(rolls, ddof=1) / np.sqrt(len(rolls)) #standard error
    interval= t.interval(0.95, len(rolls) - 1, loc=mean, scale=std_err)

    # Create histogram of roll sums
    plt.hist(rolls, bins=n_bins, edgecolor='black')
    plt.axvline(np.quantile(rolls, 0.025), linewidth = 1, color = 'red', label ='Data 95% CI')
    plt.axvline(np.quantile(rolls, 0.975), linewidth = 1, color = 'red')
    plt.axvline(mean, color='violet', linestyle='dashed', linewidth=3,label='Data mean')
    print('Mean: ', mean)
    plt.axvline(interval[0], color='green', linestyle='dashed', linewidth=2,label ='Mean 95% CI')
    plt.axvline(interval[1], color='green', linestyle='dashed', linewidth=2)
    plt.title(f"Distribution of the sum of {n_dice} dice after {n_trials} rolls")
    plt.xlabel("Sum of dice")
    plt.ylabel("Frequency")
    plt.legend()
    plt.show()

print_hist_dice(2,1000)
print_hist_dice(2,10000)
```

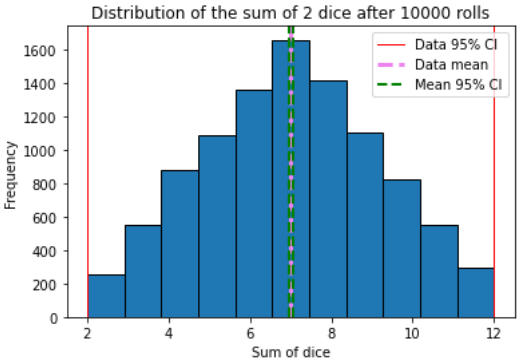
Run Code

Out [27]:

```
Mean:  6.998
```




```
Mean:  7.01
```

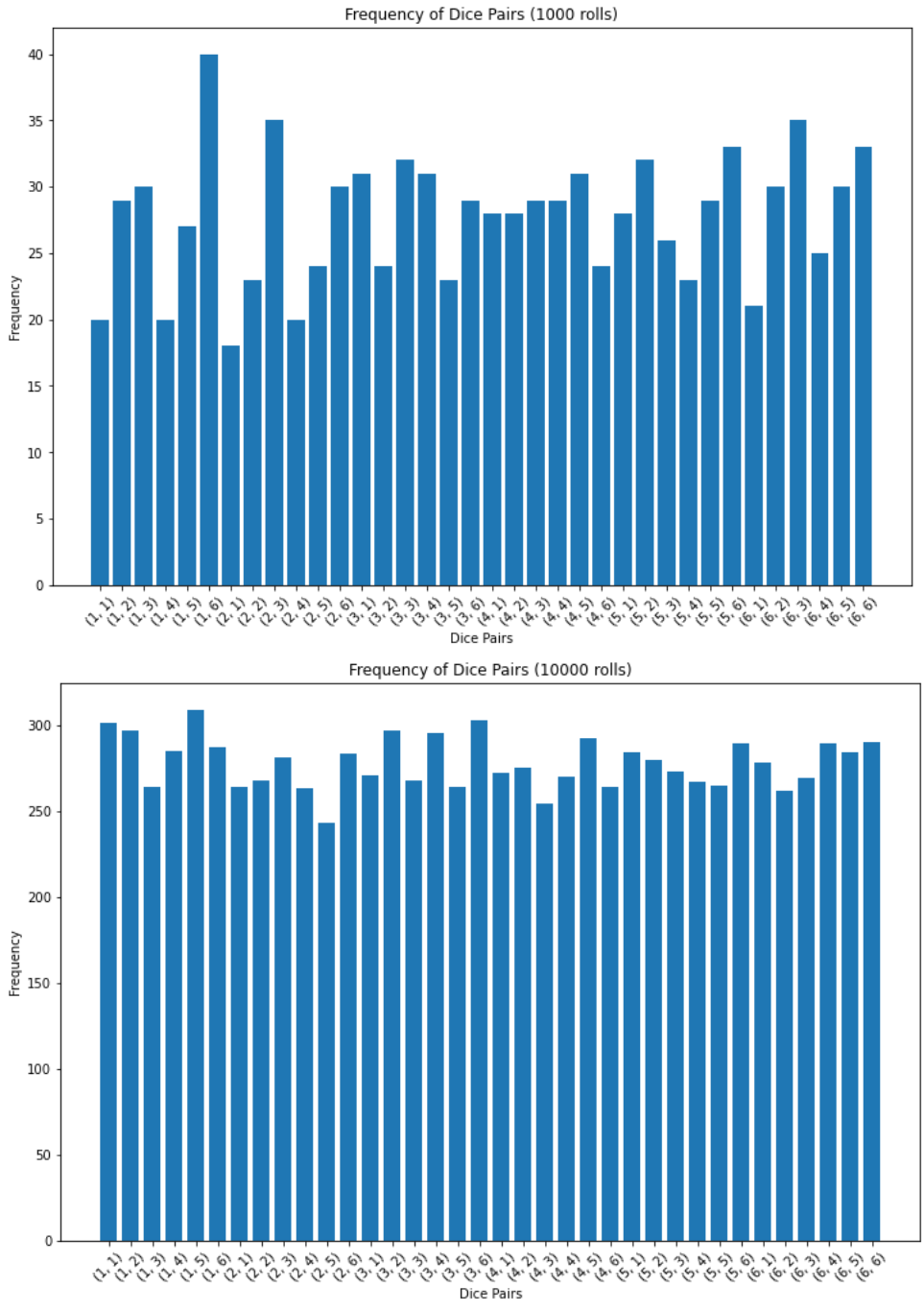


Code Cell 29 of 37

In [28]:
```
from collections import Counter
import matplotlib.pyplot as plt

def print_bar_plot_2dice(n_trials):
    '''
    The function plots the bar plot for the pairs of die rolls after a given number of trials
    ------
    Inputs:
    n_trials: int
        the number of rolls performed
    ------
    Outputs:
        a bar plot of die roll pairs for a given number of trials
    '''
    # Count the frequency of each pair
    pair_counts = dict(sorted(Counter(simulate_dice_rolls(2,n_trials)[1]).items()))

    # Extract the pairs and their corresponding frequencies
    pairs= list(pair_counts.keys())
    frequencies= list(pair_counts.values())

    x=range(len(pairs))
    ticks=[str(i) for i in pairs]
    # Create the bar chart
    plt.figure(figsize = (12, 8))
    plt.bar(x,frequencies)
    plt.xticks(x,ticks, rotation=45)
    # Add labels and title
    plt.xlabel('Dice Pairs')
    plt.ylabel('Frequency')
    plt.title(f'Frequency of Dice Pairs ({n_trials} rolls)')

    # Display the chart
    plt.show()

print_bar_plot_2dice(1000)
print_bar_plot_2dice(10000)
```

Run Code

Out [28]



Code Cell 30 of 37

```
In [28]  1
```

Run Code

Code Cell 31 of 37

```
In [29]  1 # your code here
```

Run Code

Question 9 of 11

(b) Create:

- (i) a histogram of the results for the sum of the dice for both 1,000 and 10,000 rolls; include the mean and the 95% confidence intervals for both the distribution and the mean in the corresponding histograms. If you increased the number of simulations, which confidence intervals would become narrower if any?
    a. [Optional challenge] Derive the theoretical value for the mean of the sum of dice, and compare it with the empirical results.
- (ii) a bar chart for the results of the individual pairs, and comment on the results you obtained.

Include the answers to the questions in the box provided below and use the coding cells to create the histogram and bar chart.

Normal

(i)

If we increased the number of simulations, the confidence intervals for the estimates of the mean of the distribution would become narrower.

This is because the standard error of the mean decreases as the sample size increases, which we can see from the formula for standard error in the code. The standard error of the mean is proportional to the standard deviation of the population divided by the square root of the sample size. As we increase the sample size by running more simulations, the standard error of the mean decreases, which means that the confidence intervals become narrower. This is because the larger sample size provides more information about the population distribution, which reduces the uncertainty in our estimates.

However, it's worth noting that the effect of increasing the sample size on the width of the confidence intervals will depend on the underlying distribution of the data. For example, if the population distribution has a large variance, the confidence intervals may remain relatively wide even with a large sample size. Similarly, if the sample size is already very large, further increasing it may have a diminishing effect on the width of the confidence intervals.

If we increase the number of simulations, it is unlikely that the confidence interval of the underlying data itself will change. The confidence interval is a measure of the precision of our estimate of a population parameter, such as the mean or variance, based on a sample of data. It is not a property of the population itself, but rather a function of the sample size and the variability of the data.

That is exactly what we observe in the histograms above - when we went from 1000 to 10000 trials, the CI for the mean became narrower, while the CI for the data remained the same, although there can be some variability in the results due to the stochastic nature of the simulation, especially for small number of trials.

Optional:

For 2 dice

The sum of the two dice rolled is a random variable with values between 2 and 12, with each value having a different probability of occurrence. The theoretical mean of this random variable can be calculated as follows:

Let X be the sum of the two dice rolled, then the possible values of X are {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}.

The probability of getting a certain sum x is given by the number of ways that sum can be obtained, divided by the total number of possible outcomes. The total number of possible outcomes when rolling two dice is 6^2 = 36.

The number of ways to obtain a sum of x is given by the number of combinations of two dice that add up to x. For example, there is only one way to obtain a sum of 2 (rolling a 1 on both dice), while there are two ways to obtain a sum of 3 (rolling a 1 and a 2 or rolling a 2 and a 1).

The probabilities of obtaining each sum can be calculated as follows:
P(X=2) = 1/36
P(X=3) = 2/36
P(X=4) = 3/36
P(X=5) = 4/36
P(X=6) = 5/36
P(X=7) = 6/36
P(X=8) = 5/36
P(X=9) = 4/36
P(X=10) = 3/36
P(X=11) = 2/36

The expected value or mean of X can be calculated as the weighted average of the possible values of X, with weights equal to their respective probabilities:
E(X) = 2*(1/36) + 3*(2/36) + 4*(3/36) + 5*(4/36) + 6*(5/36) + 7*(6/36) + 8*(5/36) + 9*(4/36) + 10*(3/36) + 11*(2/36) + 12*(1/36)
E(X) = 7

Therefore, the theoretical mean of the sum of two dice rolled is 7.

Comparing it to the empirical result obtained when plotting histograms above, we can see that empirical results obtained are 7.067 for 1000 rolls and 7.0262 for 10000 rolls. The empirical and theoretical results are very close to each other, suggesting their validity. Note that we we were to runt he simulation even more times, we would get the results that is even closer to the theoretical true mean, as stated by the law of larger numbers.

For 6 dice

When rolling six dice, each die can have an outcome from 1 to 6, and the sum of the six dice can take on values from 6 to 36.

To find the theoretical mean of the sum of 6 dice, we need to calculate the expected value of the sum.

Let X be the random variable representing the sum of 6 dice. Then, for any value k from 6 to 36:
P(X = k) = the probability of rolling a sum of k with six dice

To calculate this probability, we can use the fact that the number of ways to roll a sum of k with six dice is equal to the number of ways to roll k-1 with five dice, plus the number of ways to roll k-2 with five dice, plus the number of ways to roll k-3 with five dice, and so on, up to the number of ways to roll k-6 with five dice. This can be expressed mathematically as:

P(X = k) = (number of ways to roll k-1 with 5 dice + number of ways to roll k-2 with 5 dice + ... + number of ways to roll k-6 with 5 dice) / 6^6

The number of ways to roll a particular sum with 5 dice can be calculated using combinations, which gives us:
(number of ways to roll k-i with 5 dice) = (number of ways to choose i dice out of 6) * (number of ways to roll the remaining 5-i dice to add up to k-i)

This can be expressed mathematically as:
(number of ways to roll k-i with 5 dice) = C(6, i) * P(k-i, 5)
where C(6, i) is the number of combinations of 6 dice taken i at a time, and P(k-i, 5) is the number of permutations of 5 dice that add up to k-i.

Using these formulas, we can calculate the probability of rolling each possible sum with 6 dice. Once we have the probabilities, we can calculate the expected value of X as follows:
E(X) = Σ k * P(X = k)
where the sum is taken over all possible values of k from 6 to 36.

After performing the calculations, we get:
E(X) = 21

Therefore, the theoretical value for the mean of the sum of 6 dice is 21.

Comparing it to the empirical result obtained when plotting the histogram below, we can see that empirical result obtained is 21.0014 for 10000 rolls. The empirical and theoretical results are very close to each other, suggesting their validity. Note that we we were to runt he simulation even more times, we would get the results that is even closer to the theoretical true mean, as stated by the law of larger numbers.

(ii)
Analyzing the bar plot, we can see that as we rall a larger number of times (10000 instead of 1000), we get a more uniform distribution of the die roll pairs frequencies. It happens because as we rall more times, the frequency of a certain pair converges to its true theoretical frequency, which should be the same for all pairs since any pair of dice is equally likely to be rolled. This makes the bar plot uniform.

(c) Repeat the above steps (except for question 2b) for rolling six dice,  , this time simulating 10,000 rolls (please note that rolling the six dice once each corresponds to 1 roll).

Code Cell 32 of 37

```
In [30]  1 #simulating and plotting 10000 rolls of 6 dice with relevants CIs
```

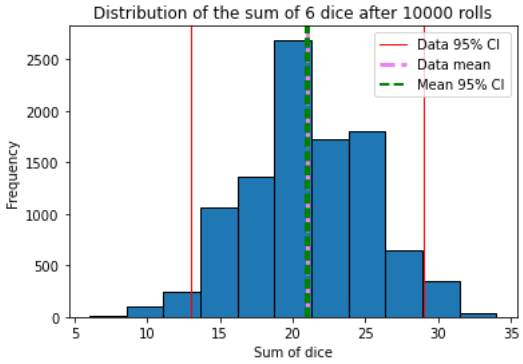
Python 3 (1GB RAM) | Edit Run All Cells Kernel Stopped |

```
2 print_hist_dice(6,10000)
```

Run Code

Out [30]

```
Mean:  21.0372
```



Code Cell 33 of 37

```
In [30] 1
```

Run Code

Code Cell 34 of 37

```
In [30] 1
```

Run Code

Code Cell 35 of 37

```
In [31] 1 # your code here
```

Run Code

Code Cell 36 of 37

```
In [32] 1 # your code here
```

Run Code

Question 10 of 11

(d) Compare the histograms for 2 dice and 6 dice when you performed 10,000 rolls and critically comment on any similarities or differences. Were the results expected? Explain.

Normal

The histograms for two dice and six dice have slightly different shapes due to the different number of possible outcomes for each case. When rolling two dice, there are 11 possible outcomes (possible sums) ranging from 2 (when both dice show 1) to 12 (when both dice show 6). However, when rolling six dice, there are 31 possible outcomes ranging from 6 (when all dice show 1) to 36 (when all dice show 6).

The difference in the number of possible outcomes for each case results in different patterns of frequencies in the histograms. In the case of two dice, the most common outcome is a sum of 7, which can be obtained in six different ways (1+6, 2+5, 3+4, 4+3, 5+2, 6+1). The frequency of each possible outcome decreases as the sum moves away from 7 as there are less ways to roll the sum 6 than 7, less way to roll the sum 5 than 6 and so on. This creates a roughly triangular-shaped histogram with a peak at 7 and roughly symmetrical around it with decreasing frequency on either side.

In the case of six dice, the most common outcome is a sum of 21, which can be obtained in 56 different ways. The most frequent sum to rall is 21, and the frequency of each possible outcome decreases as the sum moves away from 21, creating a roughly bell-shaped histogram with a peak at 21 and symmetrical around it. 46656

In summary, the difference in the number of possible outcomes for two dice and six dice results in different patterns of frequencies in the histograms. The histogram for two dice is roughly triangular-shaped with a peak at 7, while the histogram for six dice is roughly bell-shaped with a peak at 21. Although the shapes of both histograms are very similar, the histogram for the 2 dice has a less apparent peak and a wider curve than the histogram for the 6 dice due to the smaller number of possible outcomes. The probabilities of each outcome are more similar when we roll 2 dice (probability of rolling 7, most frequent, is 6/6^2 = 1/6 and probability of rolling 1, least frequent, is 1/36) than when we roll 6 dice (probability of rolling 21 is 56/6^6= 0.001 and probability of rolling 6, least frequent, is 1/6^6=0.00002). We can see that the extreme probabilities for rolling 2 dice make a much smaller range than when we roll 6 dice, resulting is a wider distribution with less sharp peak.

Overall, the results are consistent with our expectations based on the number of dice being rolled. Rolling more dice results in a distribution that is more tightly clustered around the mean with fewer extreme values (since the probabilities of getting them become very small). The histograms clearly show this difference, with the histogram for rolling six dice being more sharply peaked and less variable (also reflected in a narrower data CI) than the histogram for rolling two dice (the 95% CI often captures the extreme sums like 2 and 6).

Question 11 of 11

(e) Discuss any patterns or trends observed in the results and how they relate to the Central Limit Theorem.

Here's a code cell in case you need it.

Normal

The Central Limit Theorem (CLT) is a fundamental concept in statistics that describes the behavior of the sample variabes as the sample size increases. One of the key implications of the CLT is that, as the sample size becomes larger, the distribution of the sample variable approaches a normal distribution, regardless of the distribution of the population from which the samples are drawn.

In a die roll simulation, we can observe some patterns or trends that are consistent with the CLT. For example, when we rolled 2 fair six-sided die repeatedly, we obtained different random and independent sample variables (a sample variable would be a sum of 2 dice in one roll). As we increase the sample size from 1000 rolls to 10000, we can observe that the distribution of the sample variables (that is , sums) becomes increasingly normal.

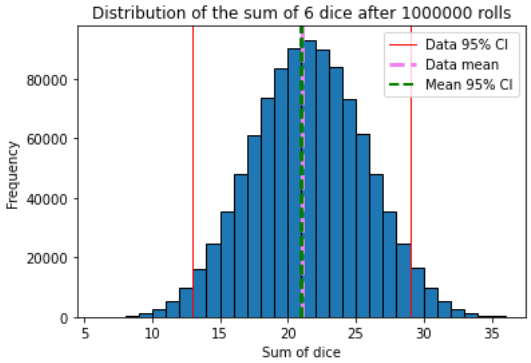
Below we can demonstrate this principle even further by rolling 6 dice more times. Comparing the histogram below (1000000 rolls) to the histograms for rolling 6 dice 1000 and 10000 times, we can see that the distribution resembles the normal distribution even more

Code Cell 37 of 37

```
In [33]  1 #simulating and plotting 1000000 rolls of 6 dice with relevants CIs
         2 print_hist_dice(6,1000000,n_bins=range(6,37))
```

Run Code

```
Out [33]    Mean:  21.004637
```


Distribution of the sum of 6 dice after 1000000 rolls
Data 95% CI
Data mean
Mean 95% CI
Frequency
Sum of dice
0
20000
40000
60000
80000
5
10
15
20
25
30
35