**DE LA SALLE UNIVERSITY - MANILA**

Realms Fate: Ultimate Test of Choices

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A Term Project

Presented to

Mr. RUIZ, RAMON STEPHEN L.

and

Mr. VELASCO, NEIL OLIVER M.

In Partial Fulfillment of the

Requirements for the PROLOGI & LBYCPA1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

by

ALFONSO, Erwin Adrian P. - EA

BELANO, Ethan Vincent Ryan B. - EB

TEJADA, Ralph Christian B. - RT

EQ2 and EQ4

4:15-5:15pm on Tuesdays and Fridays

7:30-10:30am on Mondays and Thursdays

April 20, 2023

**TABLE OF CONTENTS:**

**A. Introduction**

* 1. **Background of the Study**
  2. **Review of Related Literature**
  3. **Problem Statement**
  4. **Significance of the Project**

**B. Objectives**

**C. Methodology**

**D. Results and Discussion**

* 1. **Results**
  2. **Discussion of Results**
  3. **Output Screenshots**

**E. Conclusion**

**F. Appendices**

* 1. **User’s Manual**
  2. **Source Code**
  3. **Work breakdown**

**G. References**

1. **Introduction**

The purpose of this project is to design a well-structured program by combining various elements of topics learned throughout the semester into a single output. The project is designed to showcase the skills and knowledge acquired during the course and demonstrate the ability to integrate them into a working program.

The program will be developed in Python and will involve several key aspects such as data analysis, algorithms, and software design. The project will be divided into several stages. The first stage will involve identifying the project requirements and developing a plan for the project. The second stage will focus on data analysis and software design, followed by algorithm development in the third stage. The final stage will involve the integration of all the elements into a functional program.

The expected output of this project is a program that is well designed, efficient, and functional, meeting the requirements specified in the project specifications. This project will provide an opportunity to apply the knowledge gained throughout the course in a real-world scenario and showcase the ability to develop a high-quality program.

A Python code based game will be made. The code will be a text-based game that allows the user to make decisions and experience the consequences of their choices as the ruler of a kingdom. The code begins by initializing the starting values of four stats: people, church, army, and treasury. The user is presented with a series of scenarios, each with two options. The user must choose one of the options by inputting a number, and the program will then adjust the values of the stats based on the choice made. The scenarios include natural disasters, political turmoil, and trade negotiations.

**Background of the Study**

The aim of the project is to show and demonstrate the knowledge and skills that have been acquired throughout the course. Creating a working successful program serves as the proof of learning. The program will be coded in Python, and this will involve various elements, such as the code, the flowcharts, screenshots of the output, as well as the significance of the project, etc. The project not only aims to demonstrate the abilities acquired, but also to provide an opportunity to the audience/users that will be interacting with the project. This project will demonstrate and exercise values, which will be further explained in the significance of the project.

**REVIEW OF RELATED LITERATURE**

Reigns is a popular strategy game that was released in 2016 by Nerial. The game has gained a lot of attention for its unique gameplay mechanics and style. The main objective of the game is to maintain the balance between four different aspects of your kingdom: the church, the people, the army, and your finances. Players must make decisions by swiping left or right, which will determine the fate of their kingdom. The game has been praised for its simple yet engaging gameplay, which allows players to make quick decisions and see the consequences of their actions. The game's mechanics are designed in a way that encourages players to replay the game and experiment with different choices to see the various outcomes. Reigns' game style is a mix of medieval and modern elements, with its unique blend of humor and satire. Reigns is a well-designed game that offers a fresh and engaging experience for players who enjoy strategy games. (Hern, 2016)

The author of the game, Nerial, and publisher, Devolver Digital, are both mentioned in the article. The game is accessible for PC, iPhone, Android, and Mac users as well as other platforms. Players swipe left or right to make choices that have an impact on the game's result, which is a similar swipe-based gameplay to the first Reigns game.

The game's storyline and its integration with the Game of Thrones world are praised in the article. Various characters from the series are portrayed by players, and they are forced to make choices that will affect their rule. According to the story, players will run into characters from the series they are familiar with, and their choices will have an impact on other areas of the game. The report also mentions how well-done the game's graphics and audio are. The utilization of the series' artwork in the game and how it contributes to the story are praised in the article. The article also praises the sound design, pointing out how the music and sound effects draw players into the game's setting. The article praises the game's mechanics, storyline, graphics, and sound design and comments that it provides a distinctive and thrilling experience. (Campbell, 2018)

The popular mobile games Reigns and Reigns: Her Majesty's gameplay mechanics are thoroughly examined in the Medium post titled "Reigns, Reigns: HM and Non-Ergodicity" by Atharv Vohra. The idea of non-ergodicity—the incapacity of a system to give all participants an equal experience—is examined by the author. The article starts out by explaining non-ergodicity and how it relates to Reigns and Reigns: Her Majesty's gameplay. Because player choices greatly affect how the game turns out, the author contends that the game's mechanisms are not ergodic. Players won't have the same experience each time they play the game because each choice has a distinct outcome and there are various game endings. The mechanics of the game are then explored in detail in terms of their different non-ergodic properties. Players frequently make suboptimal decisions because they have to make judgment calls based on scant information, according to the article. The game's narrative is also non-linear, and in order to advance, players must navigate a difficult web of choices. The article also explores the game's many endings and how the player's choices affect them. The author points out that there are multiple possible outcomes in the game and that the outcomes are not preset. According to the article, this non-ergodicity is what makes the game so interesting and replayable because it encourages players to take multiple courses and consider many endings. The application of non-ergodicity to various games and systems is covered in the article's conclusion. The author points out that non-ergodicity is a characteristic of numerous games and systems and is not exclusive to Reigns. The article makes the case that non-ergodicity, which allows players to explore several paths and results, is what makes games and systems compelling and interactive. (Vohra, 2019)

In an investigation of the impact of video games on cognitive development, focusing on logical thinking and creativity. The study uses data from an online survey consisting of 10 multiple-choice questions and 39 Likert Scale questions to test the logical thinking and creativity of players and non-players of video games. Results show that on average, people who play video games tend to have higher logical thinking abilities and creativity compared to non-players. However, spending too much time playing video games negatively affects both tests. The paper examines different genres of video games and how each can stimulate different cognitive skills. For example, action games require fast-paced battles and teamwork, which can improve hand-eye coordination, reaction times, and strategic planning. Adventure games, on the other hand, require patience and problem-solving skills, while creative games boost imagination and creativity. Simulation games can improve organizational skills and logical planning, sports games enhance enthusiasts' skills, and strategy games require careful resource management. The research supports the idea that playing video games in moderation can have a positive impact on cognitive development. (IJNTR, 2015)

In addition to the cognitive benefits of learning computer coding, it also has the potential to enhance a person's non-cognitive skills. Non-cognitive skills are defined as the personal attributes and social skills that enable an individual to interact effectively with others and manage their own lives successfully. According to a study by Dede et al. (2019), coding education can improve communication skills, teamwork, perseverance, and self-efficacy among students. Through coding projects and group assignments, students learn to collaborate, communicate, and compromise to achieve a common goal. Coding also involves a high level of persistence and problem-solving skills, which can help students develop a growth mindset and increase their self-efficacy. As coding education becomes more prevalent in schools, it has the potential to not only enhance students' cognitive skills but also their non-cognitive skills, which are crucial for success in both academic and personal endeavors. (Scherer, 2021)

**Problem Statement:**

The aim of this project is to create a text-based adventure game using Python that allows the user to make decisions on how to manage a kingdom in the face of various challenges and scenarios. The program will present the user with a series of scenarios, such as a natural disaster or a rebellion, and the user must choose how to respond. Each decision will affect the state of the kingdom, including its treasury, army, church, and population.

**Significance of the Project**

This project is significant because it demonstrates our understanding of Python programming concepts and their ability to apply these concepts to a real-world problem. The game simulates decision-making in a challenging environment, which can be used to teach problem-solving and critical thinking skills. Additionally, the program can be expanded upon to incorporate more scenarios and decision points, making it a valuable learning tool for both the developer and the end-user. Overall, this project provides a practical application of Python programming skills while also contributing to the development of important life skills. At the same time, given that this project is a game, it provides the users the chance to gain better abilities in terms of creativity. In a world that is advancing rapidly especially in terms of technology, this project serves as a door or medium that can help users exercise the said values.

1. **Objectives**
2. Develop functional and efficient code: The code should be written in a clear, concise, and efficient way, with a focus on scalability, maintainability, and reusability.
3. Encourage strategic decision-making: By presenting different options with different costs and benefits, the code encourages the user to think strategically about how to respond to each scenario. This can help the user develop skills in decision-making, resource management, and risk assessment.
4. Enable customization: The code can be easily customized by modifying the initial values of the kingdom's stats or adding new scenario functions. This allows the code to be adapted for different contexts or to simulate different scenarios that might be of interest to the user.
5. Test and debug the code: Testing and debugging are essential steps in any coding project to ensure that the code is functioning correctly and to identify and fix any issues or bugs.
6. **Methodology**

Input, Process and Output Model (IPO)

Input:

The player is prompted with a scenario and given a choice of two options for how to respond.

The player inputs their choice as an integer.

Process:

The scenario functions use if/else statements to determine how to change the global variables based on the player's choice.

The global variables represent the state of the kingdom in terms of its people, church, army, and treasury.

Output:

The scenario functions print out the situation and the player's options.

The scenario functions also print out the changes made to the global variables based on the player's choice.

Output a final report of the state of the kingdom at the end of the game.

*Planning*

In this phase, we will determine the requirements of the program, such as the user interface and the functionalities that the program must have. We will also decide on the libraries required for the project.

We need to create a program that simulates a kingdom with various scenarios that the player can choose to react to. The program must keep track of the kingdom's stats, such as the number of people, the church's power, the army's strength, and the treasury's wealth.

We will use Python as our programming language for this project. No external libraries are required.

*Analysis*

In this phase, we will examine the requirements of the program and break them down into smaller modules. We will identify the inputs, processes, and outputs of each module.

For this code, we have several scenarios that the player can choose from. Each scenario has a specific input and output. The input is the player's choice, and the output is a change in one or more of the kingdom's stats.

We will create a function for each scenario that takes the player's choice as input and updates the kingdom's stats accordingly.

*Design*

In this phase, we will design the modules that we identified in the analysis phase. We will create a flowchart or pseudocode for each module to specify the logic of each function.

For each scenario, we will design a function that takes the player's input, evaluates the choice, and updates the kingdom's stats based on the outcome of the choice. We will also create a function to display the kingdom's stats after each scenario.

*Implementation*

In this phase, we will write the actual code for each module based on the design specifications. We will use Python to write the code.

We will implement the functions designed in the previous phase for each scenario, such as the player choosing to build a new church or raise taxes. We will also implement the function to display the kingdom's stats.

The main lessons taught in LBYCPA1 that were incorporated into the coding and implementation of the project are the following:

If statements- conditional statements that interpret the logic and mechanics of the game, some within and some outside of the do while loops.

Else if statements- Other conditional statements under the If statements that will implement certain logic under other conditions.

Break statements- Ends the process of the entire code as the process and mechanics of the game have been implemented and are done. It ends the process of the code so the code is not running continously.

Continue statements- It continues the code after a certain process occurs, it is used in the project to skip a specific code and to use the loop function to run the process again until a certain output is made that passes specific conditions.

Do while loops- Used to properly implement the mechanics of the game in a manner until the losing conditions are met by the user.

String Addition (Concentration)- Used to combine multiple string variables into one big string to properly display information to the User without the confusion of making multiple lines of code.

Import Random- allows an even to occur where among variables, or values a random one will be selected which is an essential step/ process for the mechanics of the game as it generates random scenarios stored in its data base.

*Testing*

In this phase, we will test the program's functionality to ensure that it is working as intended. We will run various scenarios and verify that the program's output matches the expected output.

1. Pseudocode

import random

*// initialize stats*

people = 15

church = 15

army = 15

treasury = 15

*// define scenario functions*

def dragon():

global army, treasury, church

print("A dragon has been sighted near the kingdom. What should we do?")

print("1. Send the army to fight the dragon")

print("2. Hire a group of adventurers to fight the dragon")

choice = int(input())

if choice == 1:

army -= 5

treasury -= 5

else:

church -= 5

treasury -= 5

def refugees():

global people, army, treasury

print("A group of refugees have arrived at the kingdom's borders. What should we do?")

print("1. Let them in and provide them with food and shelter")

print("2. Turn them away to avoid burdening the kingdom's resources")

choice = int(input())

if choice == 1:

people += 5

treasury -= 5

else:

people -= 5

army += 5

def trade():

global people, treasury, church, army

print("The neighboring kingdom has requested a trade agreement. What should we offer?")

print("1. Offer to trade food in exchange for gold")

print("2. Offer to trade weapons in exchange for food")

choice = int(input())

if choice == 1:

people += 3

treasury += 3

church -= 3

else:

army += 5

treasury += 5

people -= 3

def plague():

global people, church

print("A plague has broken out in the kingdom. What should we do?")

print("1. Quarantine the infected and provide medical care")

print("2. Ignore the outbreak and hope it resolves on its own")

choice = int(input())

if choice == 1:

people -= 5

church += 3

else:

people -= 10

church -= 5

def drought():

global treasury, people, church

print("A severe drought has hit the kingdom. What should we do?")

print("1. Allocate funds to import food and water")

print("2. Ration water and food, and ask the church to pray for rain")

choice = int(input())

if choice == 1:

treasury -= 10

people += 3

else:

treasury += 5

church += 5

people -= 3

def rebellion():

global army, treasury, church

print("A group of rebels have risen up against your rule. What should we do?")

print("1. Deploy the army to crush the rebellion")

print("2. Negotiate with the rebels and offer concessions")

choice = int(input())

if choice == 1:

army -= 5

treasury -= 5

else:

treasury -= 10

church += 5

def assassination():

global church, army

print("An assassination attempt on your life has been foiled. What should we do?")

print("1. Increase security and investigate the source of the attempt")

print("2. Ignore the attempt and focus on other matters")

choice = int(input())

if choice == 1:

church += 5

army -= 5

else:

church -= 3

army += 3

def flood():

global treasury, people

print("A flood has devastated the kingdom's crops and infrastructure. What should we do?")

print("1. Allocate funds to rebuild and repair")

print("2. Seek aid from neighboring kingdoms and offer to repay them in the future")

choice = int(input())

if choice == 1:

treasury -= 5

people += 5

else:

treasury += 5

people += 3

def bandit():

global army, treasury, church

print("A bandit raid has plundered the kingdom's treasury. What should we do?")

print("1. Deploy the army to capture the bandits and recover the treasure")

print("2. Offer a reward for the capture of the bandits")

choice = int(input())

if choice == 1:

army -= 5

treasury += 5

else:

treasury -= 10

church += 3

def famine():

global people, treasury, church

print("The crops have failed and the kingdom is facing a severe famine. What should we do?")

print("1. Allocate funds to import food from neighboring kingdoms")

print("2. Ration food and water, and ask the church to pray for a bountiful harvest")

choice = int(input())

if choice == 1:

treasury -= 10

people += 3

else:

church += 5

people -= 5

def earthquake():

global treasury, army, church

print("A powerful earthquake has struck the kingdom, causing widespread damage. What should we do?")

print("1. Allocate funds to rebuild damaged infrastructure and provide aid to the affected citizens")

print("2. Mobilize the army to provide aid and maintain order")

choice = int(input())

if choice == 1:

treasury -= 10

people += 5

else:

army -= 5

church += 3

def invasion():

global army, treasury, church

print("The neighboring kingdom has launched a surprise invasion. What should we do?")

print("1. Mobilize the army to repel the invasion")

print("2. Negotiate a ceasefire and offer concessions to the invading kingdom")

choice = int(input())

if choice == 1:

army -= 5

treasury -= 5

else:

treasury -= 10

church += 5

def wildfire():

global treasury, people, church

print("A massive wildfire has broken out in the kingdom, threatening crops and homes. What should we do?")

print("1. Allocate funds to contain and extinguish the fire")

print("2. Evacuate citizens and ask neighboring kingdoms for assistance")

choice = int(input())

if choice == 1:

treasury -= 5

people += 3

else:

people -= 5

church += 5

def corruption():

global treasury, church

print("Reports of corruption within the government and church have surfaced. What should we do?")

print("1. Launch an investigation and prosecute any guilty parties")

print("2. Ignore the reports and hope they will go away")

choice = int(input())

if choice == 1:

treasury -= 5

church += 5

else:

treasury += 5

church -= 5

def assassination():

global army, church

print("Reports of an assassination plot against a high-ranking official have been received. What should we do?")

print("1. Increase security and investigate the plot")

print("2. Downplay the reports and hope they are false")

choice = int(input())

if choice == 1:

army -= 5

church += 5

else:

army += 5

church -= 5

*// define scenarios list*

scenarios = [dragon, refugees, trade, plague, drought, rebellion,assassination, flood, bandit, famine, earthquake, invasion, wildfire, corruption, assassination]

*// Start display section*

print("Welcome to Realms Fate: Ultimate test of choices!\n")

print("You are the ruler of a small kingdom. Your choices will determine the fate of your people.\n")

print("Try to live as long as possible, the number of years you survived will be displayed af.\n")

print("Choose wisely!\n")

*// Initalize year count of the game*

year\_count = 0

*WHILE LOOP TRUE*

*// print current stats*

print("Current stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

*// Choose a scenario at random*

scenario = random.choice([dragon, refugees, trade, plague, drought, rebellion, assassination, flood, bandit, famine, earthquake])

*// Execute the scenario*

scenario()

*// Add one year per scenario*

year\_count += 1

*// Check the status of the existing stats if they go under or exceed a limit then loop the code then if true then exit the loop and display the users reason for death and existing stats or if the user reaches 30 years old then he or she dies of natural causes*

if people >= 30:

print("You have died! Your population has become too large and you cannot sustain them.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif people <= 0:

print("You have died! Your population has become too small and your kingdom has become vulnerable to attacks.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif church >= 30:

print("You have died! Your church has become too powerful and your kingdom has become a theocracy.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif church <= 0:

print("You have died! Your church has lost its influence and the people have lost faith in you.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif army >= 30:

print("You have died! Your army has become too powerful and you have become a military dictator.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif army <= 0:

print("You have died! Your army has been defeated and your kingdom has been conquered.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif treasury >= 30:

print("You have died! Your wealth has made you complacent and your people have suffered.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif treasury <= 0:

print("You have died! Your treasury has been looted and your kingdom is bankrupt.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif year\_count >= 30:

print("\nYou have survived for 30 years! Your rule has come to an end, You have died from natural causes.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

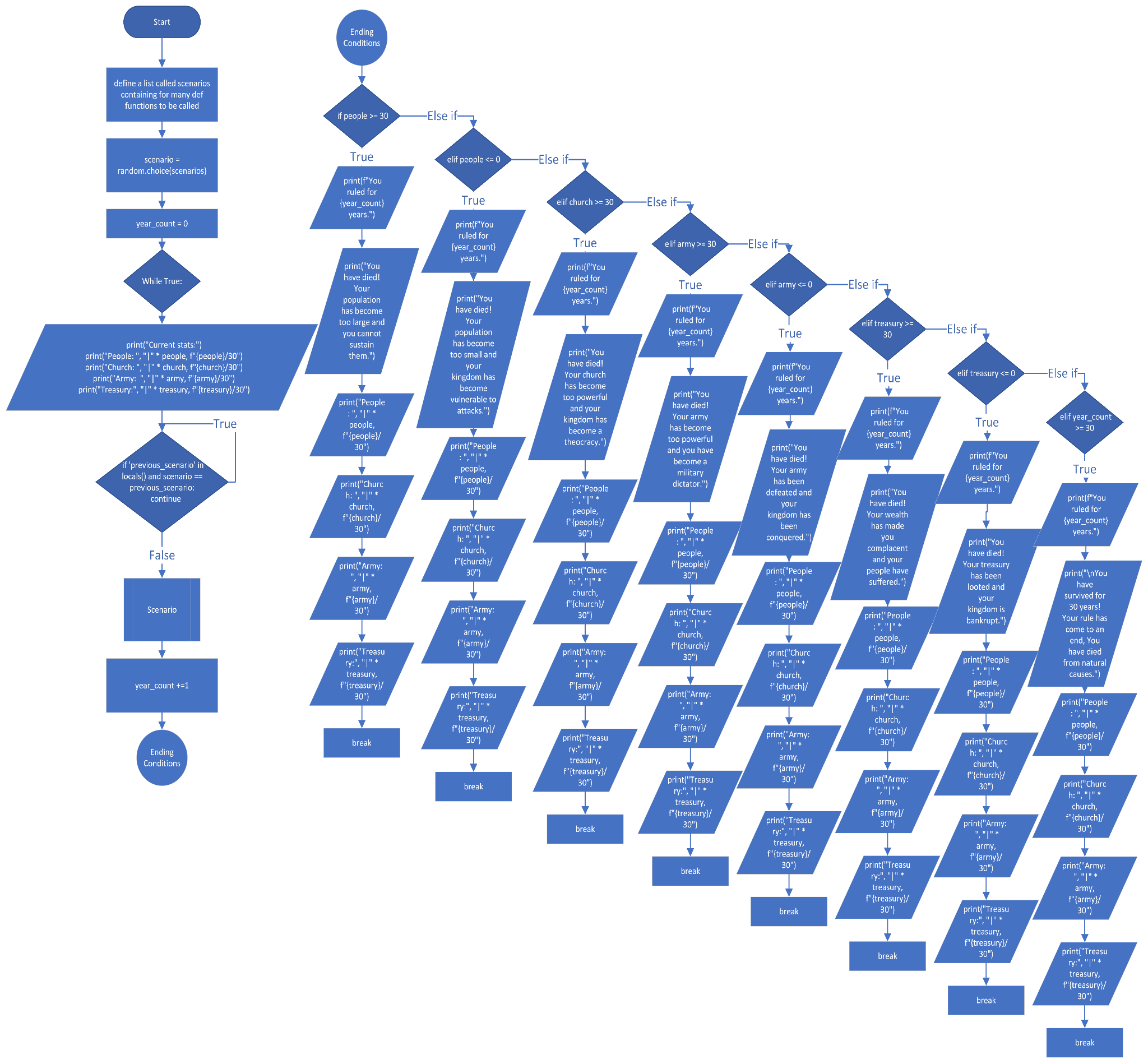
print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

*// End of code*

Conceptual Framework – IPO Chart

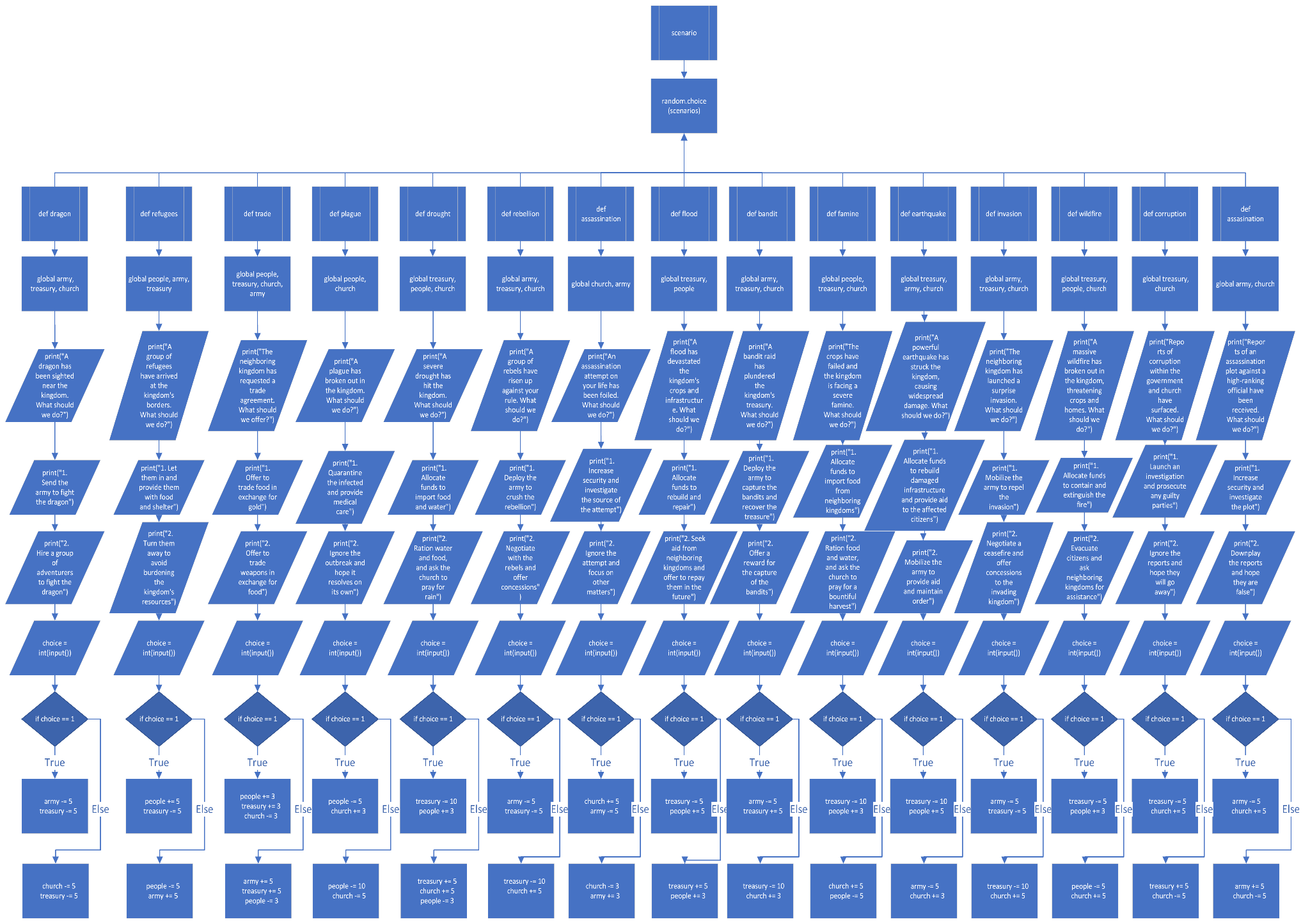
1. Flowchart



1. Hierarchy Charts for Modular Programming

Each Module in the code below is picked randomly by the main code above for a scenario that it picks as long as the same scenario is not repeated twice. The scenarios that will happen to the user vary from assassinations, famine, plagues, drought etc.

The Hierarchy chart is placed sideways to fit the document:



**Algorithm explanation for the Flowcharts above:**

Initialize the values of people, church, army, and treasury to 15.

Define scenario functions/ modules for various events, including:

dragon, refugees, trade, plague, drought, rebellion,assassination, flood, bandit, famine, earthquake, invasion, wildfire, corruption, assassination.

For each randomly chosen scenario function, prompt the user with the appropriate options 1 and 2 then update the values of people, church, army, and/or treasury based on the user's choice.

Repeat the previous steps until the values of people, church, army, and/or treasury go to 0 or to 30 and above. The game will end there as the user “dies” from his or her current reign. At the end of the game, display the final values of people, church, army, and treasury to the user. The game also ends if the number of choices or the years that the player passes reaches 30 from old age.

**D. Results and Discussion**

**Results**

The code provided functions well according to plan, with no errors occurring in the final run. The code offers a selection of random scenarios that the user can choose from, each providing a unique set of outcomes to test the user's decision-making abilities. The stats of the kingdom are displayed in an easy-to-identify way, allowing the user to track the status of the various stats with ease.

The scenarios presented in the code are well-balanced and cover a range of events that could happen in a kingdom, such as natural disasters, rebellions, and trade agreements. This provides a comprehensive range of decisions that the user can make, making the simulation more realistic and engaging.

The code's use of global variables allows for the changes in the kingdom's stats to be updated and reflected in subsequent scenarios, providing a sense of continuity and realism to the simulation.

**Discussion of Results**

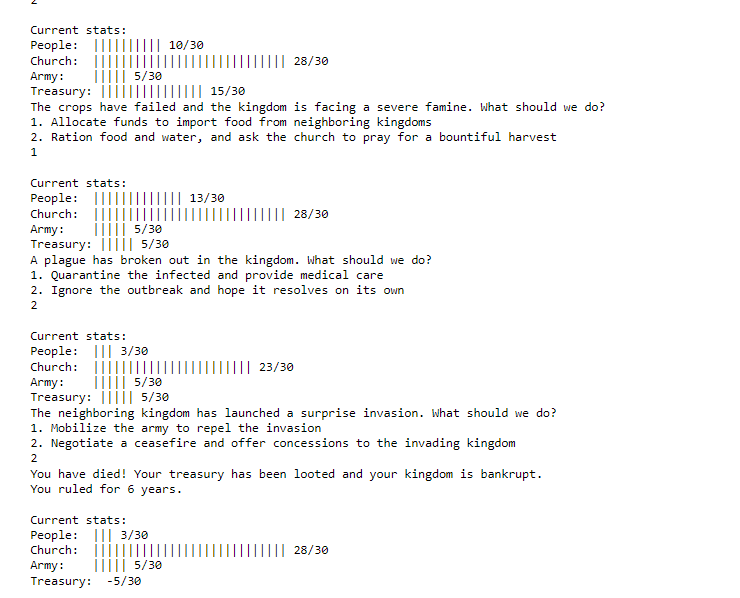
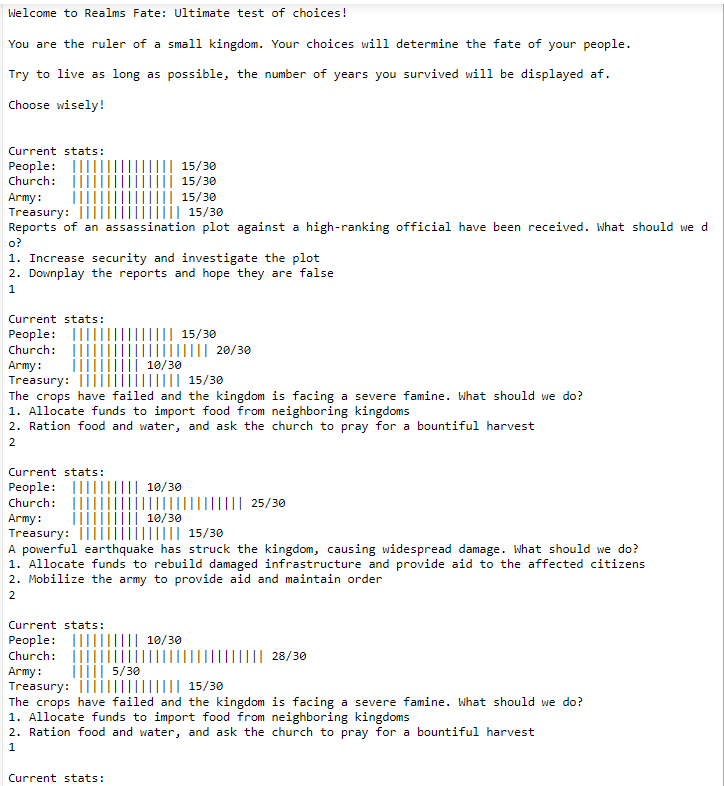
The code provided is a well-designed simulation that allows the user to make decisions as if they were running a kingdom. The selection of random scenarios and the easy-to-identify display of the kingdom's stats makes the simulation engaging and easy to use.

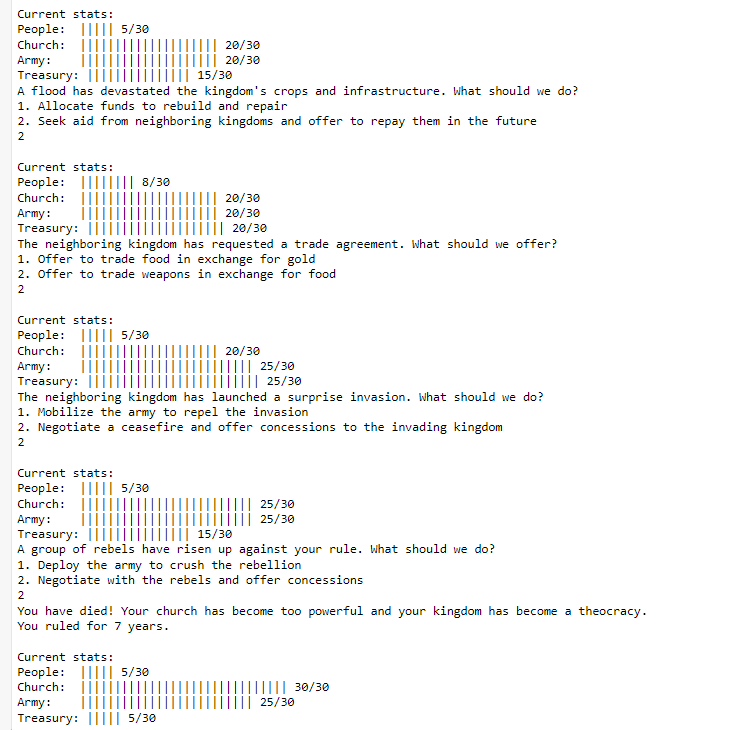
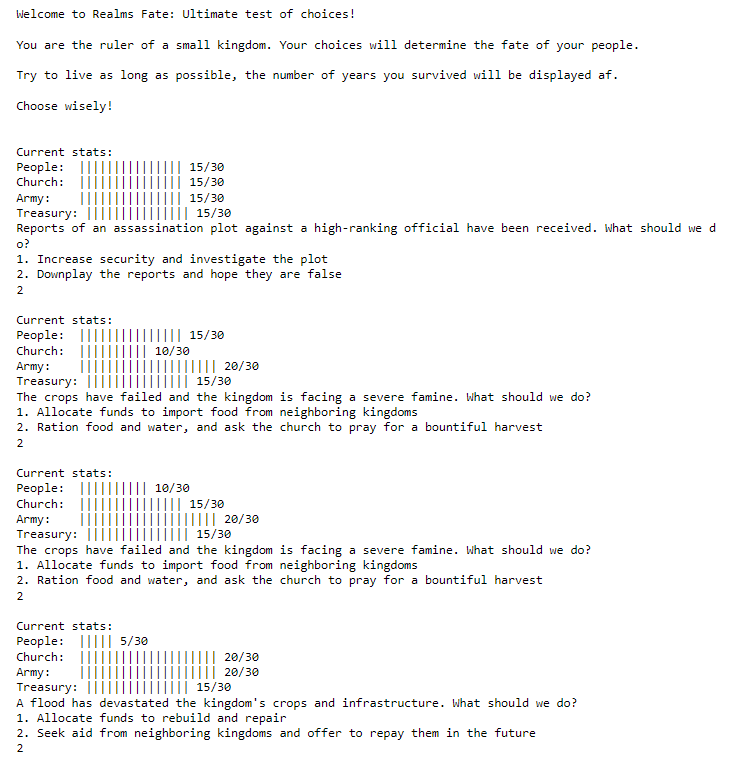
The code could benefit from a more interactive user interface, such as a graphical user interface (GUI). This would allow for a more immersive experience and provide a more visually appealing way of displaying the kingdom's stats.

Overall, the code provides a well-designed simulation that offers a range of scenarios and decisions for the user to make. With a few minor improvements, the code could be even more engaging and realistic, providing a valuable tool for teaching decision-making skills.

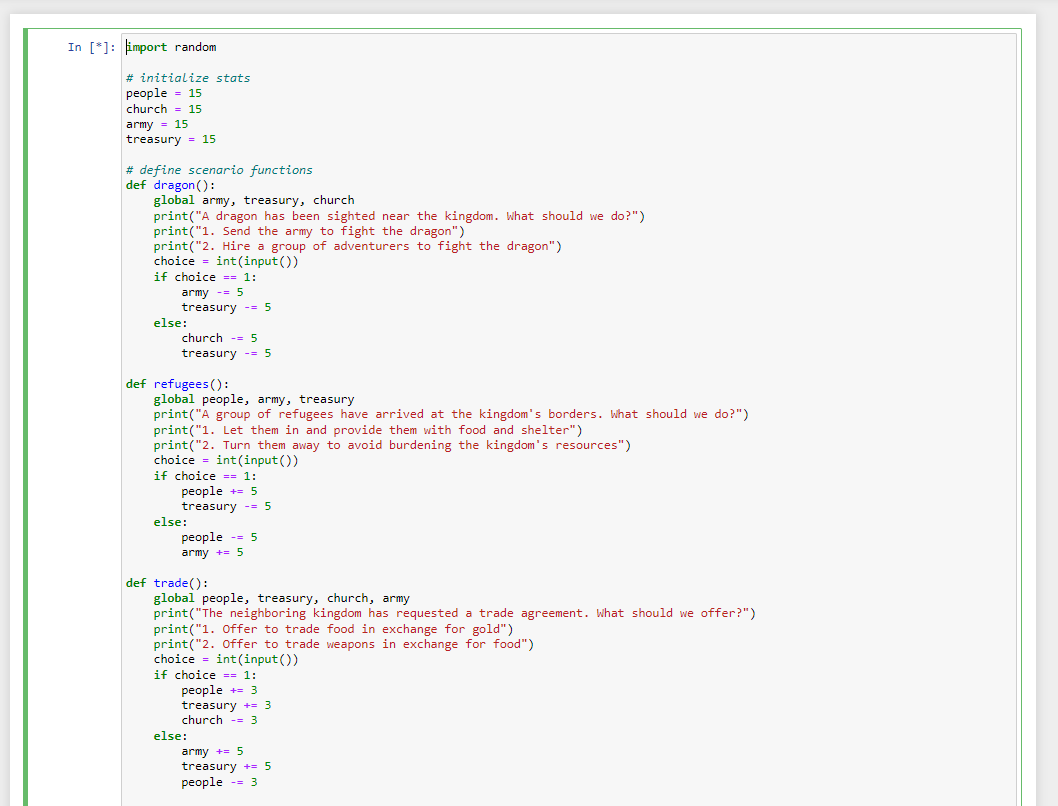
1. Sample Test Runs

Run 1



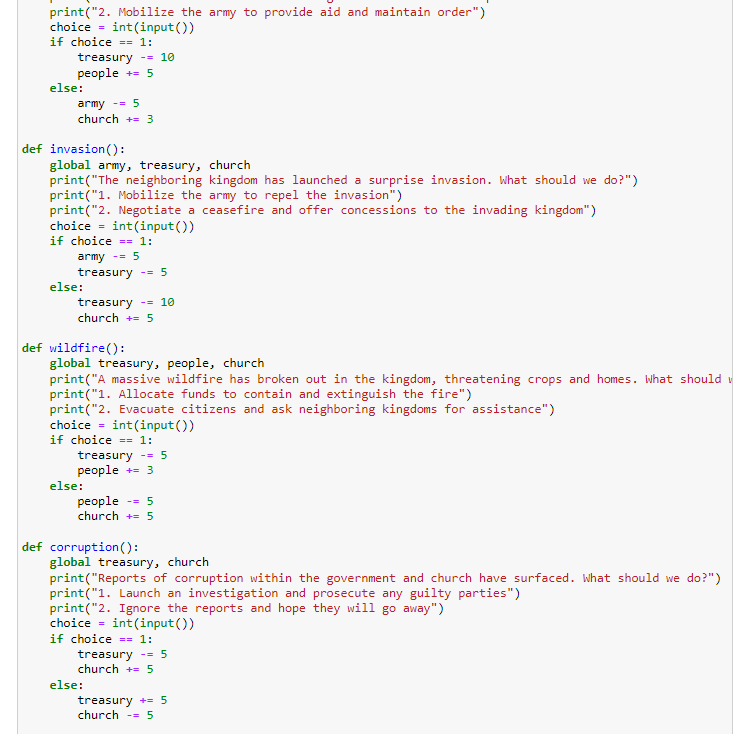
Run 2:

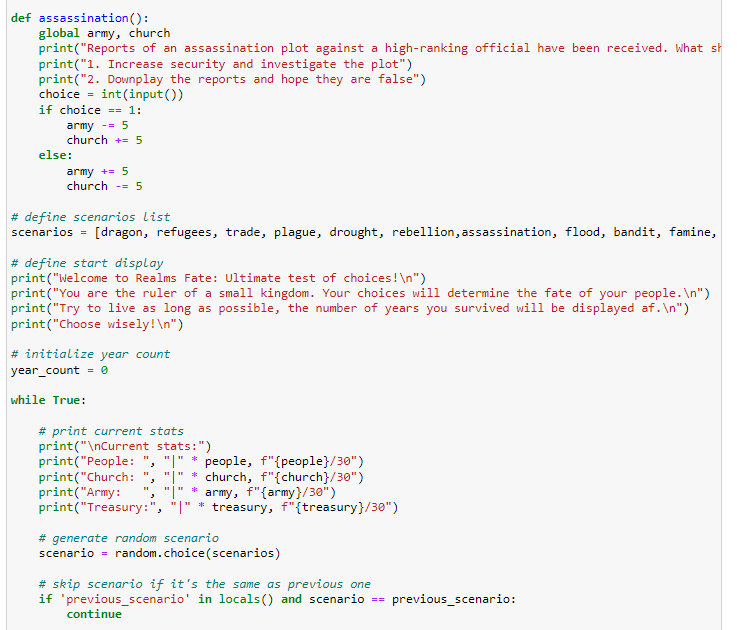
1. Different windows/screenshots of the program

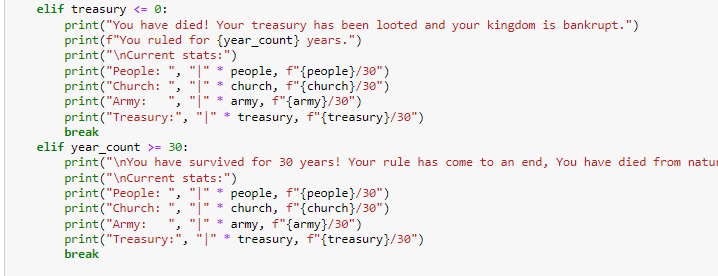
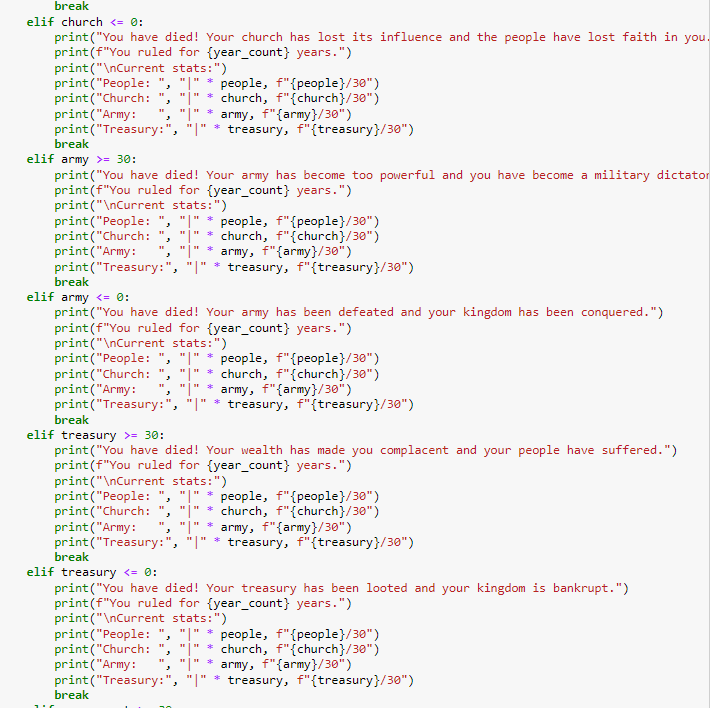












**E. Conclusion**

In conclusion, throughout the semester, we have planned, created, polished and learned coding in Python. The project we have created involves a kingdom that faces various scenarios such as natural disasters, rebellions, and trade agreements. We have used the random module to generate the different scenarios and input() function to allow the user to make choices on how to respond to each situation.

We have learned how to use global variables to keep track of the state of the kingdom and how to define functions to carry out specific scenarios. Additionally, we have learned how to use conditional statements (if-else) to handle the different choices the user can make and how to modify the variables based on these choices.

Furthermore, we have refined our coding skills by polishing the code to make it more readable and efficient. We have used comments to explain the purpose of each function and scenario, and we have used appropriate variable names to make the code easier to understand. We have also added error handling to ensure that the user inputs valid choices.

Overall, this project has been an excellent way to practice our Python skills, and we have learned how to create a functional program that simulates different scenarios based on user input. We can further expand this project by adding more scenarios, creating a graphical user interface, and implementing machine learning algorithms to predict the outcome of each scenario.

**F. Appendices**

**User’s Manual:**

To use the program read the initial instructions that appear after the code runs. The code simulates a kingdom's decision-making process when facing different scenarios. The code will instruct the user to input “1”or “2”. Each scenario affects the kingdom's stats (people, church, army, and treasury) differently. The goal is to maintain a balance between these stats and prevent any from reaching zero or from reaching 30. If the code were to be modified, the parts of the code are distributed into sectors which make the code very modular and can be modified to increase the complexity and difficulty or decrease them depending on the wish of the user.

**Source Code:**

import random

# initialize stats

people = 15

church = 15

army = 15

treasury = 15

# define scenario functions to be chosen/ randomized from later

def dragon():

global army, treasury, church

print("A dragon has been sighted near the kingdom. What should we do?")

print("1. Send the army to fight the dragon")

print("2. Hire a group of adventurers to fight the dragon")

choice = int(input())

if choice == 1:

army -= 5

treasury -= 5

else:

church -= 5

treasury -= 5

def refugees():

global people, army, treasury

print("A group of refugees have arrived at the kingdom's borders. What should we do?")

print("1. Let them in and provide them with food and shelter")

print("2. Turn them away to avoid burdening the kingdom's resources")

choice = int(input())

if choice == 1:

people += 5

treasury -= 5

else:

people -= 5

army += 5

def trade():

global people, treasury, church, army

print("The neighboring kingdom has requested a trade agreement. What should we offer?")

print("1. Offer to trade food in exchange for gold")

print("2. Offer to trade weapons in exchange for food")

choice = int(input())

if choice == 1:

people += 3

treasury += 3

church -= 3

else:

army += 5

treasury += 5

people -= 3

def plague():

global people, church

print("A plague has broken out in the kingdom. What should we do?")

print("1. Quarantine the infected and provide medical care")

print("2. Ignore the outbreak and hope it resolves on its own")

choice = int(input())

if choice == 1:

people -= 5

church += 3

else:

people -= 10

church -= 5

def drought():

global treasury, people, church

print("A severe drought has hit the kingdom. What should we do?")

print("1. Allocate funds to import food and water")

print("2. Ration water and food, and ask the church to pray for rain")

choice = int(input())

if choice == 1:

treasury -= 10

people += 3

else:

treasury += 5

church += 5

people -= 3

def rebellion():

global army, treasury, church

print("A group of rebels have risen up against your rule. What should we do?")

print("1. Deploy the army to crush the rebellion")

print("2. Negotiate with the rebels and offer concessions")

choice = int(input())

if choice == 1:

army -= 5

treasury -= 5

else:

treasury -= 10

church += 5

def assassination():

global church, army

print("An assassination attempt on your life has been foiled. What should we do?")

print("1. Increase security and investigate the source of the attempt")

print("2. Ignore the attempt and focus on other matters")

choice = int(input())

if choice == 1:

church += 5

army -= 5

else:

church -= 3

army += 3

def flood():

global treasury, people

print("A flood has devastated the kingdom's crops and infrastructure. What should we do?")

print("1. Allocate funds to rebuild and repair")

print("2. Seek aid from neighboring kingdoms and offer to repay them in the future")

choice = int(input())

if choice == 1:

treasury -= 5

people += 5

else:

treasury += 5

people += 3

def bandit():

global army, treasury, church

print("A bandit raid has plundered the kingdom's treasury. What should we do?")

print("1. Deploy the army to capture the bandits and recover the treasure")

print("2. Offer a reward for the capture of the bandits")

choice = int(input())

if choice == 1:

army -= 5

treasury += 5

else:

treasury -= 10

church += 3

def famine():

global people, treasury, church

print("The crops have failed and the kingdom is facing a severe famine. What should we do?")

print("1. Allocate funds to import food from neighboring kingdoms")

print("2. Ration food and water, and ask the church to pray for a bountiful harvest")

choice = int(input())

if choice == 1:

treasury -= 10

people += 3

else:

church += 5

people -= 5

def earthquake():

global treasury, army, church

print("A powerful earthquake has struck the kingdom, causing widespread damage. What should we do?")

print("1. Allocate funds to rebuild damaged infrastructure and provide aid to the affected citizens")

print("2. Mobilize the army to provide aid and maintain order")

choice = int(input())

if choice == 1:

treasury -= 10

people += 5

else:

army -= 5

church += 3

def invasion():

global army, treasury, church

print("The neighboring kingdom has launched a surprise invasion. What should we do?")

print("1. Mobilize the army to repel the invasion")

print("2. Negotiate a ceasefire and offer concessions to the invading kingdom")

choice = int(input())

if choice == 1:

army -= 5

treasury -= 5

else:

treasury -= 10

church += 5

def wildfire():

global treasury, people, church

print("A massive wildfire has broken out in the kingdom, threatening crops and homes. What should we do?")

print("1. Allocate funds to contain and extinguish the fire")

print("2. Evacuate citizens and ask neighboring kingdoms for assistance")

choice = int(input())

if choice == 1:

treasury -= 5

people += 3

else:

people -= 5

church += 5

def corruption():

global treasury, church

print("Reports of corruption within the government and church have surfaced. What should we do?")

print("1. Launch an investigation and prosecute any guilty parties")

print("2. Ignore the reports and hope they will go away")

choice = int(input())

if choice == 1:

treasury -= 5

church += 5

else:

treasury += 5

church -= 5

def assassination():

global army, church

print("Reports of an assassination plot against a high-ranking official have been received. What should we do?")

print("1. Increase security and investigate the plot")

print("2. Downplay the reports and hope they are false")

choice = int(input())

if choice == 1:

army -= 5

church += 5

else:

army += 5

church -= 5

# define scenarios list

scenarios = [dragon, refugees, trade, plague, drought, rebellion,assassination, flood, bandit, famine, earthquake, invasion, wildfire, corruption, assassination]

# define start display

print("Welcome to Realms Fate: Ultimate test of choices!\n")

print("You are the ruler of a small kingdom. Your choices will determine the fate of your people.\n")

print("Try to live as long as possible, the number of years you survived will be displayed after.\n")

print("Input the number of your choice and make sure the values dont exceed 30 or go below 1.\n")

print("Choose wisely!\n")

# initialize year count

year\_count = 0

while True:

# print current stats

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

# generate random scenario

scenario = random.choice(scenarios)

# skip scenario if it's the same as previous one

if 'previous\_scenario' in locals() and scenario == previous\_scenario:

continue

# execute scenario

scenario()

# increase year count

year\_count += 1

# check if any stat has gone above 30 or below 0

if people >= 30:

print("You have died! Your population has become too large and you cannot sustain them.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif people <= 0:

print("You have died! Your population has become too small and your kingdom has become vulnerable to attacks.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif church >= 30:

print("You have died! Your church has become too powerful and your kingdom has become a theocracy.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif church <= 0:

print("You have died! Your church has lost its influence and the people have lost faith in you.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif army >= 30:

print("You have died! Your army has become too powerful and you have become a military dictator.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif army <= 0:

print("You have died! Your army has been defeated and your kingdom has been conquered.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif treasury >= 30:

print("You have died! Your wealth has made you complacent and your people have suffered.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif treasury <= 0:

print("You have died! Your treasury has been looted and your kingdom is bankrupt.")

print(f"You ruled for {year\_count} years.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

elif year\_count >= 30:

print("\nYou have survived for 30 years! Your rule has come to an end, You have died from natural causes.")

print("\nCurrent stats:")

print("People: ", "|" \* people, f"{people}/30")

print("Church: ", "|" \* church, f"{church}/30")

print("Army: ", "|" \* army, f"{army}/30")

print("Treasury:", "|" \* treasury, f"{treasury}/30")

break

**Work breakdown:**

| Student Name | Tasks Assigned | Percentage of the Work  Contribution |
| --- | --- | --- |
| Erwin Alfonso | In charge of coding from defining the scenario list onwards until the break function conditional statements  Created the introduction, background of the study and methodology sector of the document | 33.33% |
| Ethan Belano | In charge of coding from the top most until defining the scenario list  Created the pseudocode  Created the significance of the project and objectives | 33.33% |
| Ralph Tejada | In charge of the related literature of the documentation  Researched about the initial codes of our project  Took initiative of the first half of the flowchart and the hierarchy chart | 33.33% |

**G. References:**

Guardian News and Media. (2016, September 7). *Reigns review: The medieval strategy game based on tinder*. The Guardian. Retrieved April 18, 2023, from <https://www.theguardian.com/technology/2016/sep/07/reigns-review-medieval-strategy-mobile-game-tinder>

*Loyalty: From single-stage loyalty to four-stage loyalty - archive*. (n.d.). Retrieved April 18, 2023, from <https://archive.org/download/IJNTR_Volume1_Issue6/IJNTR01060026.pdf>

Scherer, R., Siddiq, F., & Sánchez-Scherer, B. (2021, September 9). *Some evidence on the cognitive benefits of learning to code*. Frontiers in psychology. Retrieved April 18, 2023, from <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8458729/>

Campbell, C. (2018, October 25). Reigns: Game of Thrones review: a fun reflection on power and corruption. *Polygon*. <https://www.polygon.com/reviews/2018/10/25/18020474/reigns-game-of-thrones-review-pc-iphone-android-mac>

Vohra, A. (2021, December 9). Reigns/Reigns:HM and Non-Ergodicity - Atharv Vohra - Medium. *Medium*. <https://medium.com/@atharv.vohra/reigns-reigns-hm-and-non-ergodicity-490eed66ae33>

Rubrics for Grading the Project:

**RUBRIC FOR TERM PROJECT**

| CRITERIA | EXEMPLARY (90-100) | SATISFACTORY(80-89) | DEVELOPING (70-79) | BEGINNING (below 70) | WEIGHT |
| --- | --- | --- | --- | --- | --- |
| Experimental  Plan  (Flowchart/  Algorithm)  ***(SO-PI: B1)*** | Experimental  plan has  supporting  details and  diagram/algorithm  hm that is  stated and well  explained | Experimental  plan has  supporting  details and  diagram/algorithm  m that is stated  but not  explained | Experimental  plan is vague or  brief. It has  supporting  details and does  not have  diagram/  algorithm | No experimental plan presented | 20% |
| Codes/Data/Program | Data is well  utilized in the  program.  Program code  are easy to  read. Program  output has no  error.  Questions are  answered  completely and  correctly | Data is  somewhat  utilized in the  program.  Program code  are easy to read.  Program output  has an output  but logically  incorrect. Some  questions are  answered  completely and  correctly | Data is not  utilized in the  program. It has a  missing  significant  code/syntax in  the program | No program presented | 30% |
| Use of Appropriate Tools and Techniques **(SO-PI: K1)** | Appropriate tools and techniques are properly used for all aspects of the project | Appropriate tools and techniques are used in most of the aspects of the project and all of these are used properly | Appropriate tools and techniques are used in majority of the aspects of the project and all of these are used properly | Appropriate tools and techniques are used in less than half of the aspects of the project and/or tools are not used properly in at least half the aspects of the project | 10% |
| Project Documentation | Project documentation is orderly presented starting from statement of the problem, to objective of the project, followed by review of literature, design consideration, presentation of data or output and Conclusion. The report was grammatically correct, logically presented and used the required format. | Project documentation is complete with statement of the problem, objectives, design consideration, presentation of data and output and conclusion. The report had minimal grammatical errors and somewhat presented logically. The required format was used. | Project documentation is basically limited to algorithm presentation of data and output but no basis of the design was presented. The report had a lot of grammatical errors and not logically presented; the required format was barely used. | Project documentatio n is not reflective of algorithm design and/or characterizati on. The report had a lot of grammatical errors, was not logically presented and the required format was not used. | 30% |
| Project Presentation **SO-PI: G2** | Project presentation is complete and backed up by complete Design consideration, logic formulation and review of related literature | Project presentation is complete with algorithm simulation results backed up by design considerations | Project presentation shows a system completely simulated but is not backed up by clear explanation of how algorithm was derived | Project presentation lacks clarity in terms of presenting and characterizig the behavior of the algorithm |  |
| TOTAL | | | | | 100% |