Exercise 2

Tabular Representation of Aon Chart:

Sr.	Task	Duration	Dependencies	ES	EF	LS	LF	SLACK
1.	A	2	-	0	2	2	1	1
2.	B	1	-	0	1	2	2	0
3.	C	7	A & B	2	9	8	2	0
4.	D	5	C	9	14	18	13	4
5.	E	9	C	9	18	18	9	0
6.	F	10	C	9	19	35	25	16
7.	G	6	E	18	24	31	25	7
8.	H	7	D & E	18	25	25	18	0
9.	I	8	H	25	33	33	25	0
10.	J	2	G	24	26	33	31	7
11.	K	2	I & J	33	35	35	33	0
12.	L	2	K & F	35	37	37	35	0
							Total Slack=	35

PERT CHART & SLACK TIME:

Slack Time

ES	Task Name	EF
LS	Duration	LF

SL	=	0
0	A	2
0	2	2

SL	=	1
0	B	1
1	1	2

SL	=	0
2	C	9
2	7	9

SL	=	4
9	D	14
13	5	18

SL	=	0
9	E	18
9	9	18

SL	=	0
18	H	25
18	7	25

SL	=	7
18	G	24
25	6	31

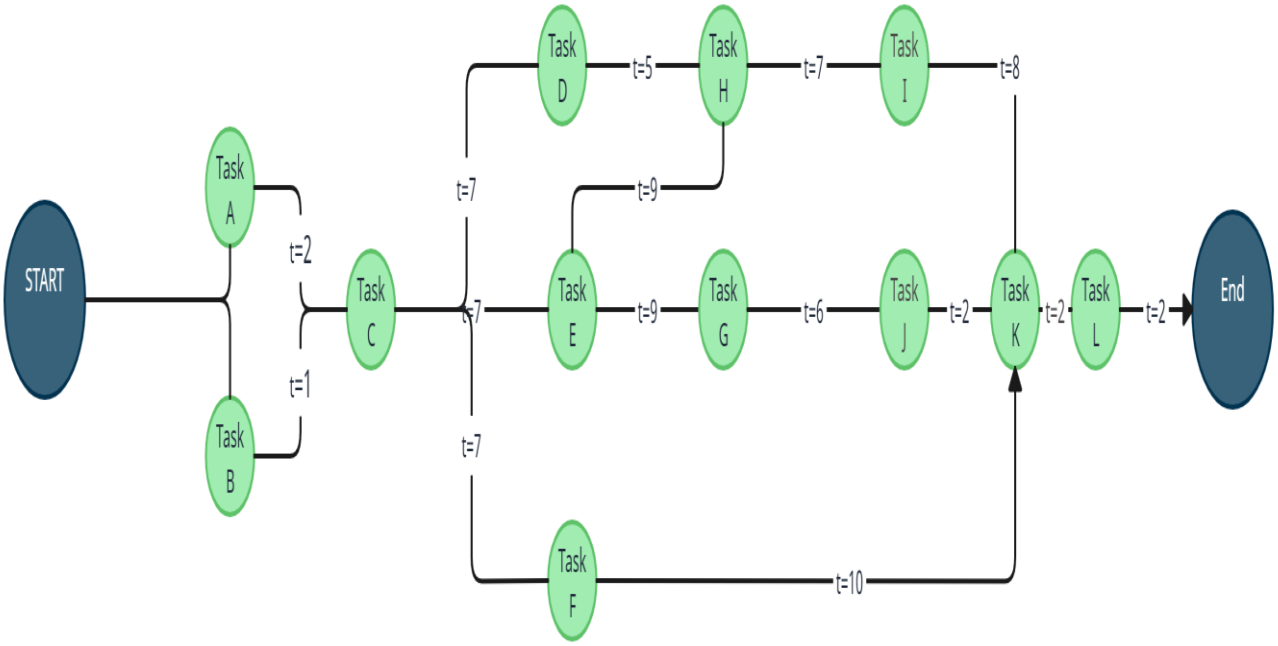
SL	=	0
25	I	33
25	8	33

SL	=	7
24	J	26
31	2	33

SL	=	0
33	K	35
33	2	35

SL	=	0
35	L	37
35	2	37

SL	=	16
9	F	19
25	10	35



Adding Activities:

if the client wants to add three activities (M, N, O) as a backlog during the sprint, the following actions should be taken:

Planning:

- The Scrum team will evaluate the new activities (M, N, O) to determine if they fit within the current sprint goal.
- The team will prioritize the new activities (M, N, O) with the existing backlog and determine their feasibility within the sprint timeline.
- The team will allocate resources to the new activities (M, N, O) based on their priority and feasibility.

Execution:

- The team will start working on the new activities (M, N, O) as part of the sprint backlog.
- The team will report the progress of the new activities (M, N, O) during the daily scrum.

Daily Scrum:

- The team will review the progress of the new activities (M, N, O) and make necessary adjustments to ensure they are delivered within the sprint timeline.
- The team will discuss any challenges or obstacles they face while working on the new activities (M, N, O) and work together to find solutions.

Review:

- The team will demonstrate the completed new activities (M, N, O) to the client and receive feedback.

Sprint Retrospective:

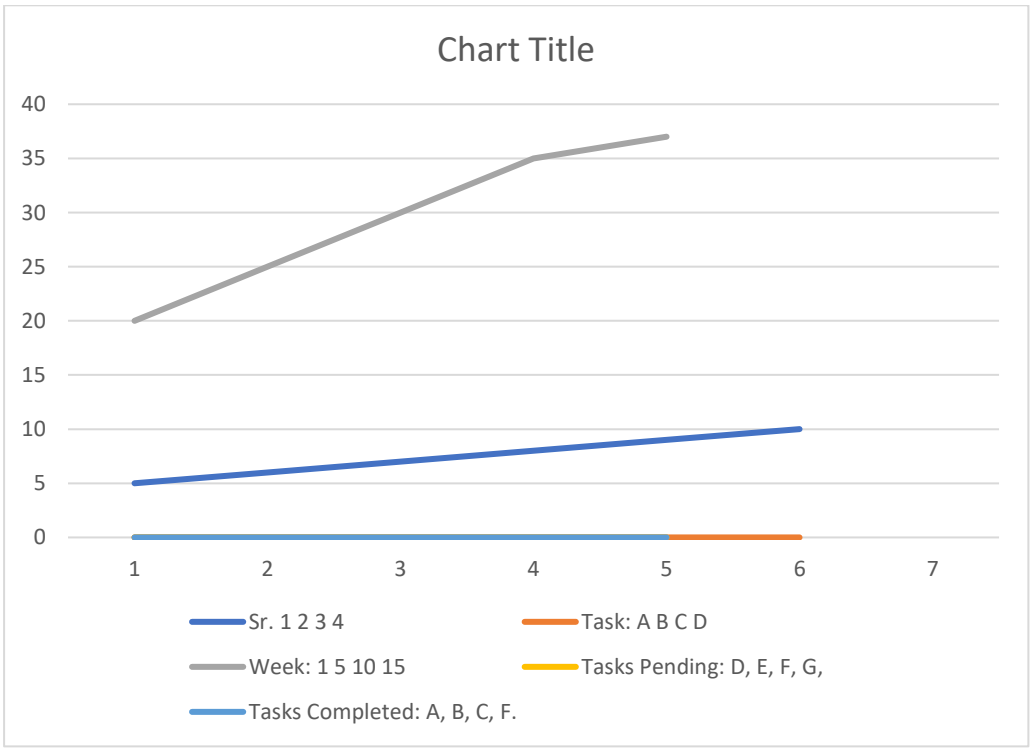
- The team will reflect on the sprint, including the impact of the new activities (M, N, O) on the sprint, and identify areas for improvement.
- The team will discuss what went well and what can be improved for future sprints.

The responsibilities of each team member in this scenario are as follows:

- The Scrum Master is responsible for facilitating the planning, execution, daily scrum, review, and sprint retrospective. They ensure that the team follows the Scrum framework and processes.
- The Development Team is responsible for delivering the new activities (M, N, O) as part of the sprint backlog. They collaborate and communicate with each other to ensure a successful sprint.
- The Product Owner is responsible for prioritizing the backlog, including the new activities (M, N, O), and ensuring that the team is working on the most valuable items. They also communicate with the client to understand their needs and expectations.

Overall, the goal in this situation is to maintain the integrity of the sprint while still accommodating the client's needs. Each team member has a specific role and responsibility to ensure the sprint is successful. The Scrum Master is responsible for facilitating the process and removing any obstacles, the Development Team is responsible for delivering the work, and the Product Owner is responsible for making decisions and managing the backlog.

Delay:



From the Burn up chart, we can see that the overall project progress is slower than the original estimated timeline, with some tasks delayed and others still pending completion. The total slack in the timeline has increased, and the overall project timeline might need to be revised to accommodate the remaining tasks and ensure the project is completed on time.

Kanban Board:

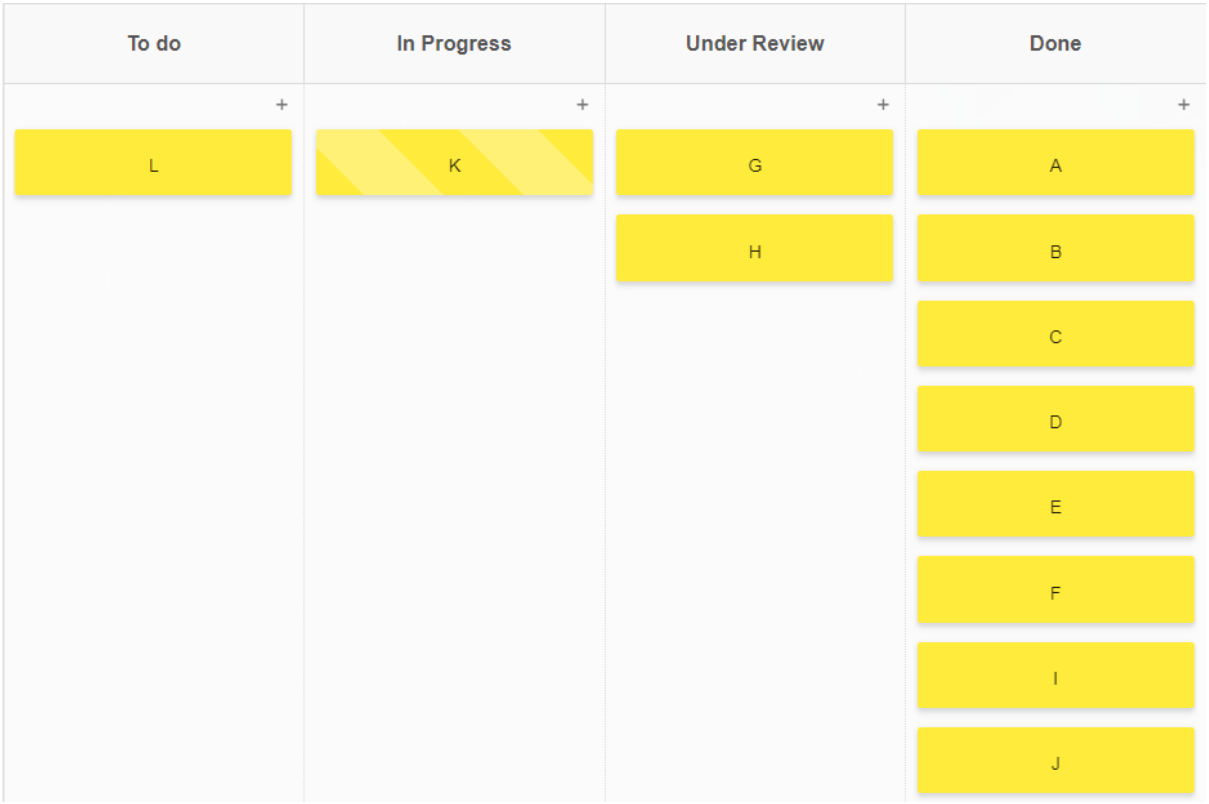


Figure 1

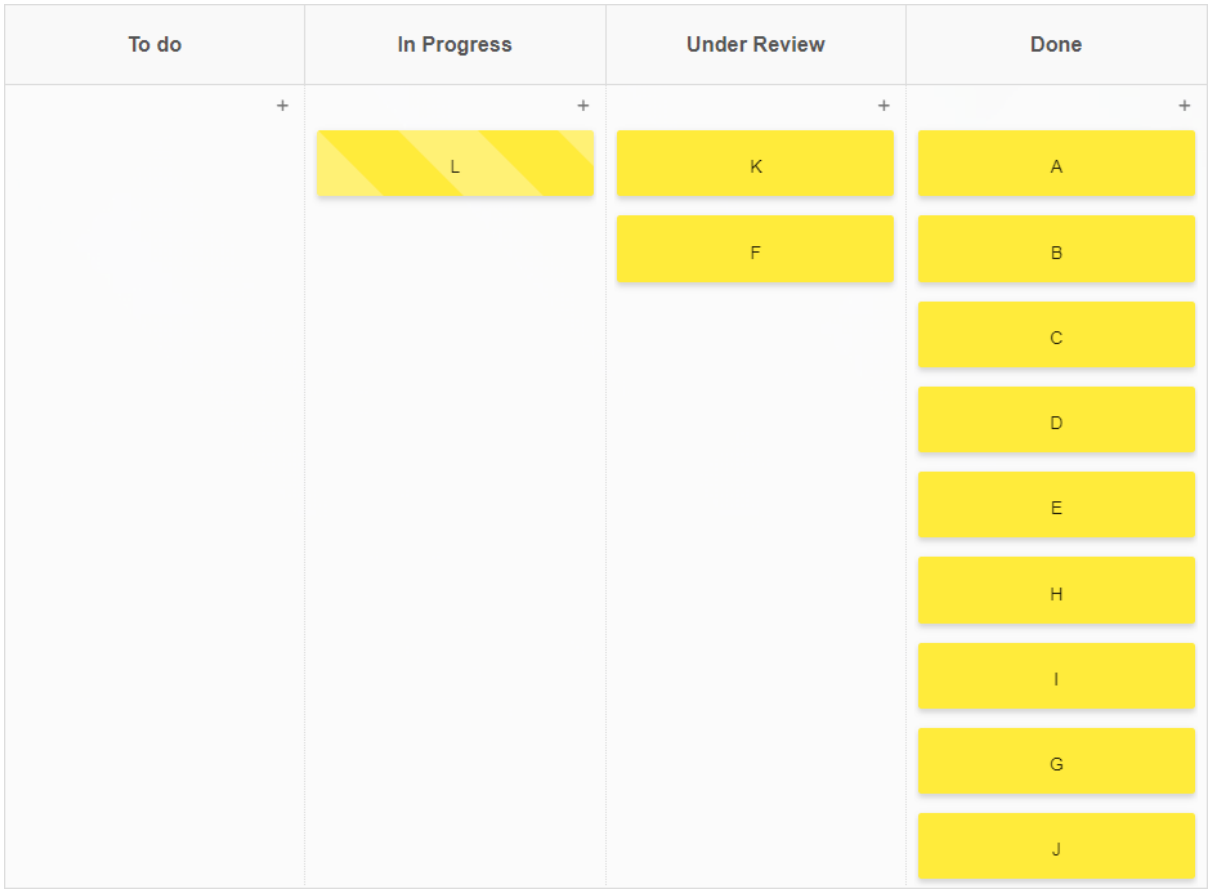


Figure 2

Functional and Non-Functional Requirements:

Vehicle manufacturing and industrial processes:

Stakeholders: Investors (human), Suppliers (human), Customers (human)

Actors: Manufacturing Machinery (non-human), Workers (human), Software (non-human)

Vehicle Manufacturing and Industrial Processes:

1. Manufacturing Machinery: "As an automated machine, I want to efficiently produce high-quality vehicles without any downtime, so that my owners can boast about their production prowess and make me the star of the factory floor."
2. Workers: "As a worker, I want to be able to control the automated machines with ease and without any hassle, so that I can spend more time playing video games and less time working."
3. Software: "As the software that runs the factory, I want to make sure all the machines are functioning optimally and the vehicles are being produced according to the specifications, so that I can be the hero behind the scenes and bask in the glory of the smooth production process."

Online application for selling homemade goods:

Stakeholders: Owner (Human), Customer (Human), Delivery personnel(Human)

Actors: Online-platform(non-human), Customer(human), Delivery personnel(human)

User Stories:

1. As an online platform, I want to provide a seamless shopping experience for customers, so they can easily browse and purchase homemade goods with just a few clicks.
2. As a Customer, I want to be able to filter search results by item type and price, so I can find exactly what I'm looking for without having to sift through a bunch of stuff I don't want.
3. As a Delivery person, I want to receive notifications when a new delivery is available and see the address and delivery instructions all in one place, so I can be the hero who brings a smile to someone's face with their homemade goods!

Use Case Descriptions:

User Story: "As an online platform, I want to provide a seamless shopping experience for customers, so they can easily browse and purchase homemade goods with just a few clicks."

Use Case Description: Title: Online Shopping Experience Primary Actor: Customer Goal: To allow customers to easily browse and purchase homemade goods with a few clicks

1.1 Basic Flow:

1. The customer opens the online platform for buying homemade goods.
2. The customer browses the goods available for sale.
3. The customer selects the desired goods and adds them to the cart.
4. The customer proceeds to checkout and enters payment information.
5. The customer confirms the order.
6. The platform confirms the order and provides the customer with an order confirmation.

1.2 Extensions:

1. The customer can filter search results by item type and price.

2. The customer can select multiple goods and add them to the cart.

3. The customer can view their order history.

User Story: "As a delivery person, I want to receive notifications when a new delivery is available and see the address and delivery instructions all in one place, so I can be the hero who brings a smile to someone's face with their homemade goods!"

Use Case Description: Title: Delivery Notifications Primary Actor: Delivery personnel Goal: To allow delivery personnel to receive notifications for new deliveries and view delivery information in one place.

2.1 Basic Flow:

1. The delivery personnel open the platform to check for new deliveries.
2. The platform sends a notification for a new delivery.
3. The delivery personnel open the notification and views the delivery information.
4. The delivery personnel deliver the goods to the specified address.
5. The platform marks the delivery as complete.

2.2 Extensions:

1 The delivery personnel can view multiple deliveries at once.

2 The delivery personnel can mark a delivery as in progress.

3 The delivery personnel can view a map with the delivery location.

Reliability Metric:

User story: "As a delivery person, I want to receive notifications when a new delivery is available and see the address and delivery instructions all in one place," the best reliability metric would be Availability.

Justification:

Availability is a measure of the percentage of time that a system is functioning and ready for use. This is relevant for this user story because the delivery person needs to be able to access the notifications and delivery information consistently and dependably, without any downtime or errors. This will ensure that the delivery person is able to perform their job efficiently and effectively, and that the customer will receive their homemade goods in a timely manner.

User story: "As an online platform, I want to provide a seamless shopping experience for customers, so they can easily browse and purchase homemade goods with just a few clicks," the best reliability metric would be MTBF (Mean Time Between Failures).

Justification:

MTBF is a measure of the average time elapsed between failures or malfunctions in a system. This is relevant for this user story because the online platform needs to be able to function consistently and dependably, without any failures or malfunctions that would disrupt the customer's shopping experience. A high MTBF will ensure that the online platform is able to provide a seamless and enjoyable shopping experience for the customer, increasing the likelihood of repeat business and customer satisfaction.

Usability requirements:

User story: "As an automated machine, I want to efficiently produce high-quality vehicles without any downtime":

a. User-friendly interface: The machinery should have a user-friendly interface that allows workers to easily operate and control the production process.

b. Customizable settings: Workers should be able to customize the settings and parameters of the machinery to suit the production requirements of different vehicles.

c. Clear error messages: The machinery should display clear error messages in case of any production issues, to help workers quickly diagnose and resolve the problem.

User story: "As a customer, I want to be able to filter search results by item type and price":

a. Intuitive search filters: The online platform should provide intuitive search filters that allow customers to easily narrow down their search results based on item type and price.

b. Clear filter options: The filter options should be clearly labeled and easy to understand, so that customers can quickly and accurately find the products they're looking for.

c. Persistent filter preferences: The online platform should remember the customer's filter preferences, so that they don't have to set them every time they visit the platform.

User story: "As a delivery person, I want to receive notifications when a new delivery is available and see the address and delivery instructions all in one place":

a. Real-time notifications: The delivery person should receive real-time notifications when a new delivery is available, so that they can quickly respond and start the delivery process.

b. Centralized delivery information: All relevant delivery information, such as the address, instructions, and order details, should be accessible in one centralized location, to reduce the risk of errors and make the delivery process more efficient.

c. Clear navigation: The delivery person should be able to easily navigate the delivery information, so that they can quickly access the information they need to complete their delivery.

Unit Testing:

Calculator Test:

```
ccadmin@CCUbuntu64bit:~/Documents/ISENFINAL$ python3 Calculator.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
ccadmin@CCUbuntu64bit:~/Documents/ISENFINAL$ vim Calculator.py
ccadmin@CCUbuntu64bit:~/Documents/ISENFINAL$ python3 Calculator.py
.FF
======================================================================
FAIL: test_percentage_num2_less_than_num1 (__main__.TestPercentage)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "Calculator.py", line 14, in test_percentage_num2_less_than_num1
    self.assertEqual(percentage(20, 10), -2)
AssertionError: -1 != -2

======================================================================
FAIL: test_percentage_valid_input (__main__.TestPercentage)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "Calculator.py", line 11, in test_percentage_valid_input
    self.assertEqual(percentage(10, 20), 0)
AssertionError: 50.0 != 0

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=2)
ccadmin@CCUbuntu64bit:~/Documents/ISENFINAL$ S
```

Core Temperature:

```
ccadmin@CCUbuntu64bit:~/Documents/ISENFINAL$ python3 CoreTempretureCalculator.py
Welcome to the reactor core warning system
1. Check reactor core temperature
2. Run tests
3. Quit
Enter an option: 2
...F.F
======================================================================
FAIL: test_reactor_core_warning_standard_temps (__main__.TestReactorCoreWarning)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "CoreTempretureCalculator.py", line 28, in test_reactor_core_warning_standard_temps
    self.assertEqual(reactor_core_warning(801), "Reactor core operating at standard temperatures")
AssertionError: 'Reactor core operating at increased temperatures' != 'Reactor core operating at standard temperatures'
- Reactor core operating at increased temperatures
?                          ^ ^ ----
+ Reactor core operating at standard temperatures
?                          ^^^ ^^


======================================================================
FAIL: test_reactor_core_warning_warning_temp_low (__main__.TestReactorCoreWarning)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "CoreTempretureCalculator.py", line 25, in test_reactor_core_warning_warning_temp_low
    self.assertEqual(reactor_core_warning(666), "Warning! Core temp low. Decreased efficiency.")
AssertionError: 'Reactor core operating at standard temperatures' != 'Warning! Core temp low. Decreased efficiency.'
- Reactor core operating at standard temperatures
+ Warning! Core temp low. Decreased efficiency.


----------------------------------------------------------------------
Ran 6 tests in 0.001s

FAILED (failures=2)
Tests complete.
```

Guessing Game:

```
ccadmin@CCUbuntu64bit:~/Documents/ISENFINAL$ python3 GuessingGame.py
FGuess a number between 1 and 100: 40
Too low
Guess again: 50
Too high
Guess again: 42
You got it!
.
======================================================================
FAIL: test_median_three_numbers (__main__.TestFindMedian)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "GuessingGame.py", line 12, in test_median_three_numbers
    self.assertEqual(find_median(3, 2, 1), 1) # this test fails
AssertionError: 2 != 1

----------------------------------------------------------------------
Ran 2 tests in 6.856s

FAILED (failures=1)
ccadmin@CCUbuntu64bit:~/Documents/ISENFINAL$ vim  GuessingGame.py
```

Modularity:

Function Name	**Identified Issue**	**Category of Issue**	**Comments on Issue**	**Proposed Solution**

get_player_beast	Hardcoded input prompt	Input Prompt Hardcoding	The input prompt for choosing the beast is hardcoded with player number	The player number is passed as an argument to the function, allowing the input prompt to be dynamically generated based on the player number
attack	Hardcoded dice type and number	Hardcoded Game Mechanics	The dice type and number used in the game attack mechanics are hardcoded, making it difficult to change these values	The dice type and number are passed as arguments to the function, allowing them to be easily changed if needed
play_game	Health values hardcoded	Hardcoded Game Mechanics	The starting health values of the beasts are hardcoded, making it difficult to change these values	The starting health values are defined as variables, allowing them to be easily changed if needed
---	Inconsistent variable naming	Variable Naming	The variable names used in the code are inconsistent, making it difficult to understand the code	The variable names have been made more consistent throughout the code, making it easier to understand

The input prompt for the player to choose their beast was hardcoded as "Player 1, choose your beast: " or "Player 2, choose your beast: ". This made it difficult to reuse the code for different games or player numbers. The updated code solves this issue by passing the player number as an argument to the function, allowing the input prompt to be dynamically generated based on the player number.

The dice type and number used in the attack mechanics were also hardcoded in the code, making it difficult to change these values. The updated code solves this issue by passing the dice type and number as arguments to the attack function, allowing them to be easily changed if needed.

In the code, the starting health values of the beasts were hardcoded as 50. This made it difficult to change these values if needed. The updated code solves this issue by defining the starting health values as variables, allowing them to be easily changed if needed.

The variable names used were inconsistent, making it difficult to understand the code. The updated code solves this issue by making the variable names more consistent throughout the code, making it easier to understand.

Flags:

In the original code, the flag "is_defending" is used in the "attack" function to determine the outcome of the attack. If the flag is set to True, the function returns 0, meaning the player is

defending and no damage is dealt. However, using flags like this in code can affect modularity in several ways:

1. Code Complexity: The use of flags can make the code more complex and harder to understand, as the logic behind the flags is not immediately apparent.
2. Coupled Logic: The flag and the logic that it controls are tightly coupled, meaning that if the logic behind the flag changes, the flag itself also needs to be changed, making it difficult to modify the code without introducing errors.
3. Hard to Reuse: Using flags in functions can also make it harder to reuse the function in different parts of the code, as the flag needs to be set correctly for the function to work as expected.

In the updated code, the issue with the flag is addressed by breaking down the "attack" function into separate functions that each handle a specific aspect of the game logic. This makes the code easier to understand, more modular, and more reusable.

Ethics and Professionalism:

A Mobile Phone GPS Maps App - Project Management:

Project management. In the development of this type of app, proper project management is crucial. If the project manager does not keep track of deadlines, resources, and team progress, it could lead to a harmful outcome. For example, if the project is not properly managed, the app may not be released on time. Users who were excited to use the app may become frustrated and lose trust in. This could lead to harm.

To avoid this outcome, it is important for the project manager to maintain clear communication, set realistic deadlines, and effectively utilize the resources available to ensure the successful and timely release of the app.

B. AI Research project- Testing:

Testing is a crucial aspect of the software engineering process. It ensures that the AI system is working as intended and meets the necessary requirements and standards. However, if proper testing procedures are not followed and the necessary care is not taken, the outcome can be harmful.

A lack of thorough testing can result in the AI system producing inaccurate results, which can lead to unintended consequences. For example, if the AI is being used for medical diagnosis, a lack of proper testing can result in incorrect diagnoses and treatments, causing harm to patients. This could lead to serious consequences, such as misdiagnosis, incorrect treatment, and even death.

Therefore, it is important that the testing process for AI Research projects is well-planned, executed, and thoroughly reviewed to minimize the risk of harmful outcomes. This requires the development team to have a high level of professionalism and to follow best practices in testing software systems.

C. Online Retail Store: -Area of software engineering:

Functional requirements -Harmful outcome: A lack of proper understanding of the functional requirements for the online store can result in a poor user experience. For example, if the website is not user-friendly or if it has a slow loading time, customers are likely to become frustrated and go to a competitor's website, resulting in a loss of revenue for the company. This can result in a user experience that is less than satisfactory.

For example, if the website is difficult to navigate, customers may have trouble finding the products they are looking for. If the website loads slowly, customers may become impatient and go to a competitor's website instead. It is important for software engineers to have a thorough understanding of the functional requirements in order to create an online retail store that meets customer needs and expectations.