Mini-Project Report On

Merchant of Venice (A Visual novel Game)

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

By

Ankit John Abraham (U2003037)
Alwin Shibu (U2003030)
Christopher Jacob (U2003061)
Ashwin Saji (U2003220)

Under the guidance of
Ms. Jisha Mary Jose

Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039

CERTIFICATE

This is to certify that the mini-project report entitled ***"Merchant of Venice (A Visual Novel Game)"*** *is a bonafide work done by* ***Mr. Ankit John Abraham (U2003037), Mr. Alwin Shibu (U2003030), Mr. Christopher Jacob (U2003061), Mr. Ashwin Saji (U2003220)****, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2022-2023.*

Dr. Preetha K. G.	**Dr. Sminu Izudheen**	**Ms. Jisha Mary Jose**
Head of Department	Mini-Project Coordinator	Mini-Project Guide
Dept. of CSE	Asst. Professor	Asst. Professor
RSET	Dept. of CSE	Dept. of CSE
	RSET	RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal of RSET, and **Dr. Preetha K. G.**, Head of Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Merchant Of Venice(A Visual Novel Game)".

We are highly indebted to our mini-project coordinators, **Dr. Renu Mary Daniel**, Assistant Professor, Department of Computer Science and Engineering, and **Dr. Sminu Izudheen**, Assistant Professor, Department of Computer Science and Engineering for their valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Ms. Jisha Mary Jose**, for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Ankit John Abraham

Alwin Shibu

Ashwin Saji

Christopher Jacob

ABSTRACT

The challenge is to create a Visual Novel Game that presents a fresh adaptation of the classic tale by William Shakespeare, "The Merchant of Venice".This interactive storytelling experience combines striking visuals, captivating music, and dynamic branching choices, allowing players to actively engage with the narrative and influence its outcomes.The game centers around the intertwining lives of three central characters: Antonio, the generous merchant; Shylock, the vengeful moneylender; and Portia, a clever and resourceful heiress. Players embark on a journey through the vibrant and bustling streets of 16th-century Venice, exploring its richly detailed environments through beautiful AI generated pictures that evoke the era's art style.

The methodology in which we would create a project of this scale and size includes, but is not limited to a, comprehensive literature review of the original text by William Shakespeare, story boarding and outlining the Visual novel's key scenes,asset creation through prompt engineering using generative AI such as DALL-E,Stable Diffusion,Leonardo AI,etc, programming and implementing the story and art assets as an actual visual novel game and last but not least playtesting and ensuring optimum quality of the game's mechanics. The final game would then be run either from the user's personal computer as an executable application or through their web browser as an online game.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

1.1.1 Growing Aversion to Reading in children

The increasing use of electronic gadgets, such as smartphones, tablets, and computers, has led to a decline in reading behavior among children.According to a study published by the National Literacy Trust as part of a World Book Day, shows that in 2019 just 26 percent of under-18s spent some time each day reading. This is the lowest daily level recorded since the charity first surveyed children's reading habits in 2005.

Several Factors may have contributed to this phenomenon such as the instant gratification and Visual Stimulation earned through the usage of electronic gadgets,A shorter attention due to the ever increasing pace of the world,Lack of supervision from their parents or Social pressure.

1.1.2 Visual Novels as an alternative to Books

"Books are the quietest and most constant of friends; they are the most accessible and wisest of counselors, and the most patient of teachers." That may have been true until the 20th century when the book became more accessible for the common people and the reading was the window to the world and a way of entertainment as well as education. However, in the 21st century, with the development of the technology and telecommunications, the Internet has become the new window to the world. With the development of the electronic book readers, tablets and other similar devices, the books are slowly being adapted and/or published in electronic version, so that they can be read on the computer or using these devices. Still the youth are more interested in multimedia games, which have animation, visual effects, sounds and music.It is here that Visual Novels come into

prevalence.

The origins of the visual novel date from the 1980s with the game The Poftopia Serial Murder Case [6] (jap. or Poffopia Renzoku Satsujin Jiken), which was first published in the Japanese market in 1983 by the company Enix (today's Square Enix). The genre of this game was adventure and it was made for the Japanese computer NEC PC-6001. Later in 1985, the company Chunsoft adapted the game for NES (Nintendo Enteffainment System) and this version of the game has become precursor of the visual novels. However, the company Chunsoft published the first novel in 1992 and it was the horror thriller game Otogirisö (jap. eng. St John's won) which was made for Super Nintendo Enteffainment System. This game, with 300 000 copies sold, was so successful that it became a role model for the new genre and the games where named as Sound Novels (jap. F or saundo noberu). One specific characteristic of these games is that after the first play, with new staff and new decisions made, a new additional scenario would be unlocked. The company Leaf, which was one of the many that wanted to develop this type of games, in order not to use Chunsoft's trademark Sound Novel, named their series games Visual Novels. Leaps first games were not so successful until 1997, when they published the game To Healt (jap. or Tu Häto). With its warm and touching love stoy, as well as its high quality music, it became instant success. This motivated other companies to accept this term and since then these types of games are known as Visual Novels.

Telling stories through Visual Novels allows players to:

- Have an interactive experience
- Enhance the reader's imagination
- Explore different types of stories
- Enjoy stories at their own pace
- Improve Comprehension and retention
- Experience a sense of exploration and replayability
- Enjoy the thrill of gaming while not being as inclined towards reading

1.1.3 The Merchant of Venice By William Shakespeare

We have decided to tell the classic tale Merchant of Venice by William Shakespeare as an interactive visual novel game.

William Shakespeare's "The Merchant of Venice" is a timeless and captivating play, widely regarded as one of his most complex and thought-provoking works. First performed around 1596-1598, the play takes place in the richly atmospheric and bustling city of Venice, where the destinies of its diverse cast of characters intertwine against a backdrop of love, revenge, justice, and prejudice.

At its core, "The Merchant of Venice" explores themes that remain relevant even to this day. The play delves into the complexities of human nature, shedding light on the nature of friendship, the power of love, and the destructive force of hatred and intolerance. Moreover, Shakespeare's exploration of the legal system and the concept of justice adds a layer of social and moral commentary to the narrative.

1.2 Existing System

- Printed Books: You can read physical copies of "The Merchant of Venice" by purchasing printed books from bookstores, online retailers, or borrowing them from libraries.

- E-books: "The Merchant of Venice" is available in digital formats as e-books. You can find these e-books on various platforms such as Amazon Kindle, Apple Books, Google Play Books, and other e-book retailers.

- Online Texts: Many websites offer free access to the full text of "The Merchant of Venice" online. You can read the play directly from these websites without any cost.

- Audiobooks: If you prefer listening to the play rather than reading it, you can find audiobook versions of "The Merchant of Venice" on platforms like Audible, Librivox, and other audiobook providers.

- Educational Platforms: "The Merchant of Venice" may be available on educational platforms that focus on classic literature. These platforms might offer additional

annotations and analysis to help readers better understand the text.

- Anthologies and Collected Works: Shakespeare's works, including "The Merchant of Venice," are often included in anthologies or collected works of his plays. These editions might have additional contextual information and analysis.

- Mobile Apps: There are mobile applications dedicated to Shakespeare's works, where you can find "The Merchant of Venice" alongside other plays and sonnets.

- Scholarly Publications: Academic journals and literary publications often analyze and discuss Shakespeare's plays, including "The Merchant of Venice." Reading such analyses can deepen your understanding of the play.

- School or University Resources: If you are studying "The Merchant of Venice" as part of a school or university course, your educational institution may provide access to textbooks or online resources that include the play.

1.3 Problem Statement

The challenge is to tell the classic tale 'Merchant Of Venice' by William Shakespeare as an interactive visual novel game.

One looking to read the story may be faced with several difficulties such as unfamiliarity with Shakespeare's use of early modern English-Archaic words and expressions,Unfamiliar terminology,Puns and wordplay,etc.The reader may also face difficulty in confronting themes of prejudice and anti-semitism as well as lack the time and patience of reading a play of such density and length.

The application addresses these problems by simplifying the text and making it interactive by telling the story as a visual novel game.

1.4 Objectives

- **Storyboarding**- To simplify the complex archaic english used by william shakespeare into something more understandable by adults and kids alike and developing branching choices for this linear storyline to be used in the visual novel game.

- **Image Asset Generation** - To create various image assets such as those for the main characters,locations and other items through prompt engineering using generative AI such as DALL-E,Stable Diffusion,Leonardo AI,DreamWeaver AI,etc.

- **Audio Asset Generation** -To find audio assets that would match different scenes in the game from free sound libraries such as SoundBible or other sources.

- **Script to rpy conversion** - To convert the script written in English language to the engine's scripting language to be saved in a file with .rpy extension so that in the final game, the character appears to be speaking.

- **Mini Games and Puzzles**- To add make the game more challenging and interesting by adding puzzles or minigames that use mathematical and engineering principles along with the story.

1.5 Scope

The visual novel game will cover the main plot of "The Merchant of Venice," focusing on the intertwined lives of Antonio, Shylock, and Portia. It will explore key events such as the bond between Antonio and Shylock, the trial scene, and the casket test. While some aspects of the original play may be condensed or adapted to fit the interactive format, the narrative will remain faithful to Shakespeare's vision.

Depending on the success and feedback received, future iterations of the visual novel game could potentially expand the story, explore additional character arcs, and include further adaptations of Shakespearean plays.

The game may also be further improved by adding a text-to-speech module so that the characters actually speak the dialogues displayed on the screen.

Chapter 2

Literature Review

In this literature review, we explore the integration of classic literature into a visual novel game, along with the inclusion of various puzzles. Previous studies have highlighted the potential benefits and challenges of merging timeless literary works with interactive gameplay. The combination of rich literary content and puzzles presents an opportunity to engage players in both storytelling and problem-solving activities.

2.1 Serious Games and Educational Value in the Context of "Merchant of Venice"

An examination of the potential educational value of using classic literature as the foundation for a visual novel game with puzzles. Studies such as in paper [8] have shown that serious games, those designed with a purpose beyond entertainment, can enhance learning outcomes. By incorporating puzzles related to the themes and plots of classic literary works, the visual novel game may offer students a unique and interactive way to explore the content while developing critical thinking and problem-solving skills.

2.2 Puzzle Design and Player Engagement in the Visual Novel Game

This section delves into the design and impact of four puzzles in the visual novel game inspired by classic literature. Research shown in paper [9] suggests that serious games, including puzzle-solving elements, can positively influence student motivation, particularly when traditional learning methods may not appeal to all students. Investigating how these specific puzzles enhance player engagement within the context of classic literature can shed light on their effectiveness in reinforcing the themes and messages of these literary works.

2.3 Effectiveness of the Visual Novel Game for Literary Appreciation and Comprehension

Explore the effectiveness of the visual novel game in improving students' understanding and appreciation of classic literature compared to traditional teaching methods. Studies mentioned in paper [10] show that game-based learning and literacy development is particularly relevant in this context. By incorporating elements of the narratives, characters, and themes into the gameplay, the visual novel game has the potential to facilitate deeper connections and meaning-making for students as they interact with the stories through the puzzles.

2.4 Impact of Puzzle Difficulty on Player Learning and Experience

Assess the impact of puzzle difficulty levels on player learning and overall gaming experience. The challenges may vary in complexity, and the level of difficulty could influence players' engagement and satisfaction. Drawing insights from Sichter's (2016) research [7] on the impact of game narratives on critical thinking, we can examine how puzzle difficulty affects players' ability to comprehend and interpret classic literature.

2.5 Cultural Adaptation and Representation in the Visual Novel Game

Examine the adaptation of classic literature in the visual novel game and the representation of cultural themes and characters. As De Cavallaro et al. [2] highlighted, the integration of classic literary works into an interactive medium brings challenges in maintaining the essence of the original texts. Analyzing how the visual novel game handles cultural nuances and themes can offer valuable insights into the effectiveness of using timeless literature in modern gaming formats.

2.6 Educational Applications of the Visual Novel Game for Literature and Puzzle-based Learning

Discuss the potential applications of the visual novel game in educational settings, particularly for literature and puzzle-based learning. Drawing on the findings from various studies, including Iten and Petko [4], Garneli et al. [9], and O'Donnell [10], we can assess

the viability of using similar game formats to engage students in literary exploration and critical thinking.

2.7 Conclusion

In conclusion, the integration of classic literature in a visual novel game with puzzles presents an innovative approach to engaging students in timeless literary works while promoting problem-solving and critical thinking skills. The literature review provides a comprehensive analysis of the potential benefits and challenges of this integration, paving the way for further research and development in the field of interactive literature education.

Chapter 3

System Analysis

3.1 Expected System Requirements

The system of user which is a personal computer or smart phone is expected to have the following features:
Personal Computer:

- Windows platform with version 7.0 or above.
- Requirement of Internet connection for the web version of the game.
- A storage space of approximate 100 MB for the app.
- A minimum Ram size of 1 GB is required in the device.
- Nvidia,Intel or AMD GPU with OpenGL 2 or WebGL support

Smartphone:

- Android platform with a version 5 or above.
- Requirement of Internet connection for the web version of the game.
- A storage space of approximate 100 MB for the app.
- A minimum Ram size of 1 GB is required in the device.
- Adreno or Mali GPU with OpenGL ES 2 or WebGL support

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

The project is technically feasible since majority of the population are in possession of smartphones and personal computers. The app only requires minimum requirements to run on these devices.

3.2.2 Operational Feasibility

The operations are built in a simple and easy to use manner for adults and kids alike.Running the app from it's executable file or visiting our website to play it directly in the browser is the only prerequisite operation to be done.

3.2.3 Economic Feasibility

The app can reduce the time and effort needed by people to read and understand the complex story line of 'The Merchant of Venice' by William Shakespeare.It may also promote reading behaviour in children as it is an interactive visual novel game.The development of the app is also zero budget as it was built using free resources.

3.3 Hardware Requirements

The following are the system requirements to develop the Visual Novel Game.

- Processor: Intel Core2 Duo E8400 or above,AMD Athlon 64 or above
- Hard Disk: Minimum 80GB
- RAM: Minimum 4 GB

3.4 Software Requirements

The following are the software used in the development of the app.

Operating System: Windows or Linux

3.4.1 Ren'Py Visual Novel Engine

Ren'Py is a free and open-source visual novel engine that enables creators to develop interactive storytelling experiences. It was created by Tom "PyTom" Rothamel and first released in 2004. Ren'Py is written in the Python programming language and is designed to be accessible to both beginners and experienced developers, making it a popular choice for creating visual novels and other narrative-driven games. Some of the key features of Ren'Py for Visual Novel development include:

- Scripting Language: Ren'Py uses a simple and easy-to-learn scripting language based on Python. This language allows developers to define the game's story, characters, dialogues, and choices.

- Visual Novel Elements: Ren'Py provides built-in support for visual novel features, such as character sprites, background images, music, sound effects, and text boxes. This simplifies the process of creating visual novel games by providing a framework for common elements.

- Choice-based Interactivity: Ren'Py allows developers to create branching narratives with multiple choices, where player decisions influence the direction and outcome of the story. This interactive element adds depth and replayability to the games.

- Cross-Platform Support: Ren'Py games can be deployed on various platforms, including Windows, macOS, Linux, and Android. This allows creators to reach a wide audience on different devices.

- Built-in Text Editor: Ren'Py includes a built-in text editor that makes it easy to write and edit the game's script directly within the engine.

- Theming and Customization: Creators can customize the appearance of their visual novels by defining themes, changing fonts, and modifying the user interface elements.

- Community and Resources: Ren'Py has an active community of developers and enthusiasts who share resources, tutorials, and extensions, making it easier for newcomers to learn and improve their game development skills.

- Commercial Use: Ren'Py allows for commercial game development, enabling creators to release their visual novels commercially without any licensing fees.

Ren'Py has been used to develop a wide range of visual novels, interactive fiction, and narrative-driven games across different genres, including romance, mystery, horror, and fantasy. Its user-friendly nature and versatility have contributed to its popularity among independent game developers and hobbyists alike.

Overall, Ren'Py provides a powerful and accessible tool-set for storytellers to create engaging and interactive narratives in the form of visual novels, offering an enjoyable experience for players and readers alike.

3.4.2 Leonardo AI

Leonardo AI is a generative AI platform that empowers users to effortlessly generate captivating images and artwork, with a specific emphasis on game assets.

With a focus on accessibility, the platform offers free usage with limited daily tokens, allowing users to explore their artistic capabilities without financial barriers. By customizing image generation costs based on quality and quantity, Leonardo AI offers a flexible and affordable solution for digital artists.

Leonardo achieves it's results by using the deep-learning model Stable Diffusion.However, one can use their own training data to create their own personalised AI model.

3.4.3 Adobe Photoshop

Adobe Photoshop is a powerful and widely-used raster graphics editing software developed by Adobe Inc. It was first created by Thomas Knoll and John Knoll in 1987 and has since become the industry standard for image editing, manipulation, and graphic design.Photoshop supports features such as Image editing,Layers,Selection tools,Filters and Effects,Typography,Content-Aware tools,Image format support etc.

Chapter 4

Methodology

4.1 Proposed Method

Developing a visual novel game based on William Shakespeare's "Merchant of Venice" using interactive visuals, music, and puzzles in Ren'Py requires careful planning and execution. Here's a step-by-step methodology along with a block diagram explaining each block of the development process:

1. Project Planning:
    - Define the scope and objectives of the visual novel.
    - Identify the target audience and the platform(s) for deployment.
    - Allocate resources, including team members, time, and budget.
    - Create a project schedule with milestones and deadlines.
2. Story Adaptation:
    - Understand the original story thoroughly.
    - Decide which parts of the story to include and how to present them.
    - Develop a script that adapts the play into a visual novel format.
    - Consider incorporating multiple story paths or endings to add replay value.
3. Art and Design:
    - Create character designs and backgrounds that fit the game's setting and theme.
    - Design the user interface (UI) for menus, dialogue boxes, and interactive elements.

- Develop concept art and storyboards to visualize the scenes and sequences.

4. Music and Audio:

   - Compose or source music and sound effects that complement the game's atmosphere.
   - Design the user interface (UI) for menus, dialogue boxes, and interactive elements.
   - Ensure the audio quality is suitable for the game and enhances the player's experience.

5. Programming in Ren'Py:

   - Familiarize yourself with Ren'Py and its scripting language.
   - Implement the game's branching narrative using Ren'Py's visual novel engine.
   - Integrate music and audio into the game at appropriate moments.
   - Create puzzles or interactive elements using Ren'Py's functionality.

6. Block Diagram Explanation:

   - Visual Assets: This block represents all the art assets required for the game, including character sprites, background images, and UI elements.
   - Audio Assets: Here, you store the music tracks, sound effects, and voiceovers (if any) that will enhance the game's atmosphere.
   - Scripting & Programming: This block covers the coding aspect of the game, where you use Ren'Py's scripting language to implement the story, branching paths, puzzles, and interactive elements.
   - Game Engine (Ren'Py): Ren'Py is the game engine used to bring all the elements together. It handles the presentation of visual assets, audio playback, and script execution.
   - Puzzles & Interactive Elements: This block refers to the parts of the game where players engage with puzzles, make choices, or interact with the story.

- Testing & Quality Assurance (QA): In this phase, the game is thoroughly tested to identify and fix bugs, ensure a smooth user experience, and refine the gameplay.
- Deployment & Distribution: Once the game is complete, it is packaged for distribution on the chosen platforms, such as PC, Mac, or mobile devices.
- Player Feedback & Iteration: This block represents an essential ongoing process. After release, player feedback is collected, and updates or patches may be developed to address issues or enhance the game further.

7. Testing and Feedback:
    - Conduct extensive testing to identify bugs, typos, and any other issues.
    - Gather feedback from playtesters to understand their experience and make improvements accordingly.
8. Final Polishing:
    - Address any remaining issues from testing and feedback.
    - Fine-tune the gameplay, visuals, and audio to ensure a polished and cohesive experience.
9. Release and Promotion:
    - Prepare the game for release on the chosen platforms.
    - Implement a marketing strategy to promote the game and reach the target audience.

Chapter 5

System Design

5.1 Architecture Diagram

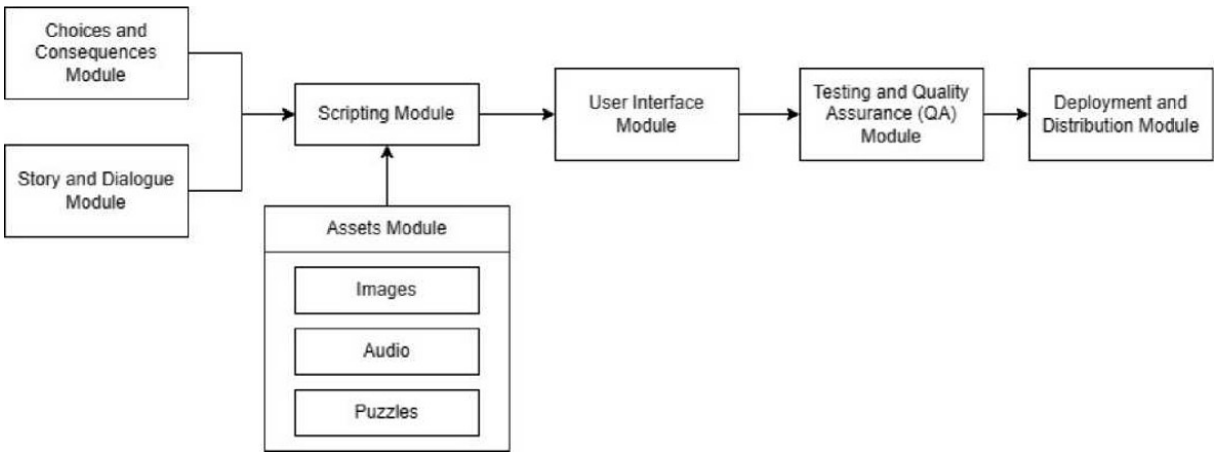

Figure 5.1: Architecture diagram

The architecture diagram shows how the various modules in the game interact with each other.First a comprehensive script of the game is developed by adding the story,it's choices and consequences along with the images and audio assets and also the puzzles or mini games are called when needed.They are then presented to the user in dynamic graphical user interface so that the user can interact with the game through the use of textbuttons or imagebuttons to do various functions such as pausing the game, saving the state ,loading a saved state ,interacting with the puzzles and so on.The game is then tested to ensure that the various puzzles,branching choices and the flow of the story is ensured.The final game is then built and deployed on a platform of our choices using the build functionality in the Ren'py engine

5.2 Sequence diagram

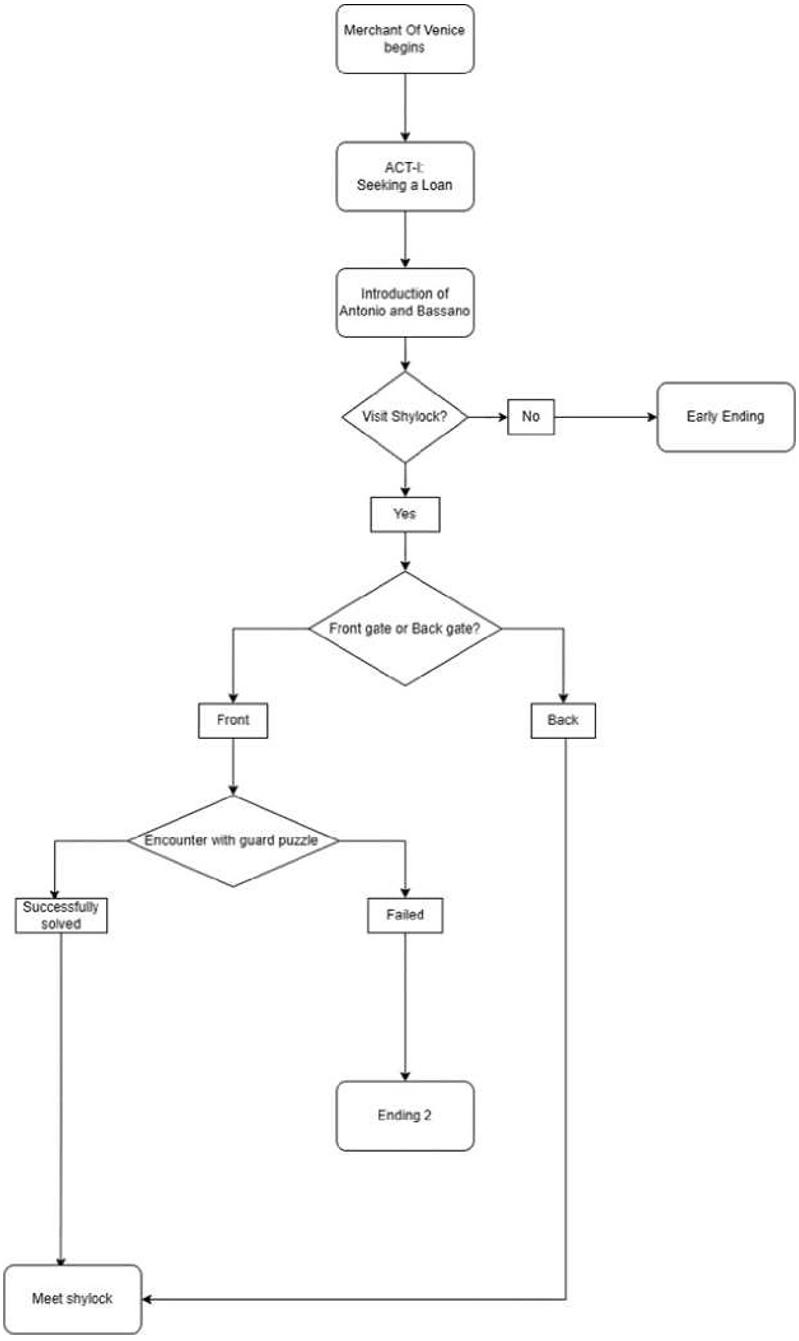

Figure 5.2: Sequence Flow Diagram

The above diagram shows an example sequence-flow of various events in an alpha version of the first level of the game.The game presents the user with various branching choices early on in the narrative some leading to early endings.It is evident that the user only encounters puzzles if they have chosen that branch in the story.Such a level of complexity allows dynamic narratives in the game.

Chapter 6

System Implementation

6.1 Ren'Py Scripting

The English language script is converted to a code that is recognised by the engine. The characters are first defined using the keyword 'define' and then called using their identifier whenever it is their turn to speak.The code for different levels in the game are separated into different rpy files for ease of use.The levels and branching choices are defined using the keyword 'label' and called using jump statements only when the user reaches that particular point in the game or if the user chooses that particular branch in the story.This type of abstraction ensures that the general flow of the story is established.

The module also deals with the decision logic according to which the dialogues in the game are displayed.At certain points in the game, the user may be presented with weighted choices which add up when the user chooses one among them.The cumulative weight is then taken and compared using simple if-else statements to create branches in the story.

6.2 User Interface

This module deals with the placement and style of several UI elements that are used in the game.GUI elements are developed as png or jpg files for the various screens in the game such as the main-menu,pause-menu,saving and loading screens as well as for textboxes,menu bars, etc using photoshop and stored in a 'gui' folder in the game's directory.These image files are then arranged in their desired position by defining their x and y coordinates using a combination of two files-screens.rpy and gui.rpy.

6.3 Sprites & Animations

The game makes use sprites and animations to give life to its characters.Character sprites generated either in jpg or png format are stored in an 'images' folder within the game's directory.The characters are then shown and hidden on the screen according to the narrative using the 'show' and 'hide' commands.The show command takes the character's identifier as input to show the character at any particular (x,y) coordinate on the screen, which can be further tweaked using offset values whereas the hide command simply hides the character from the screen.These commands can be further tweaked by using animations to give movement to the character.Some notable animations available within Ren'Py are move,moveinleft,moveinright and dissolve.

6.4 Puzzles

The game makes use of puzzles and minigames to add additional challenge to the narrative.The first notable puzzle that the player may encounter is a tower of hanoi puzzle.Here,the player is asked to solve the classic tower of hanoi problem with 3 poles and 4 disks in the minimum number of moves which is 15 inorder to progress further in the story.

This puzzle is implemented in way that at that point in the story a jump statement is called in order to go to a different label where the puzzle logic is defined.When it comes to the game's logic, first the starting pole and destination pole are randomly set using a random function in python.Then a screen which sets up the layout of the game is called which constructs three poles along with the disks stacked in descending order on the starting pole.The image assets used in the game are stored in the same 'images' folder along with the game's character sprites.The screen makes use of vboxes or vertical boxes to display various textbuttons such as an option for the user to give up ,an option for the user to see the minimum number of moves needed ,the arrow keys that determine to movement of the disks as well as an undo button for the player to revert his actions by using a for loop that enumerates through each of these towers.While the screen remains clickable the different cases that may arise for each of these towers include

- Case 1-The player hasn't decided yet what move to make and there are some disks

on the pole: Here the '↑' arrow key is displayed below that particular pole indicating the player that he can move disks from that pole onto another.The variable start_from_tower is set as 'i' in case the player clicks on this arrow

- Case 2-The player has decided to move from a particular pole: This case is checked by verifying whether the variable start_from_tower is equal to i.In this case a textbutton 'undo' is is displayed below the pole from which the player has selected his disk indicating that he can keep back the taken disk on it's original pole.When this textbutton is pressed the variable start_from_tower is set as -1

- Case 3-The disk that the player picked is smaller than the topmost disk on a pole: This case is verified if either towers[i]["blocks"] is an empty list or if towers[start_from_tower]["blocks"][0]["size"] is less than towers[i]["blocks"][0]["size"].Here the '↓' arrow key is displayed below that particular pole indicating that the player can place his disk on that particular pole.Upon pressing that arrow key, a variable 'finish_to_tower' is set as 'i' and a value 'move_done' is returned to the calling function.

- Case 4-The disk can't be moved onto the pole: This case occurs if the any one the above three conditions are not satisfied.Here an empty string " " is displayed below the pole with no functionality.

If screen interactions are disabled then the game performs similarly as case 4.The player's move count is actively tracked throughout the duration of the game and is incremented when making any one of these moves.The game is only won if the player's move count is equal to the minimal number of moves for that particular number of disks.After that the player is allowed to proceed forward in the story otherwise he is shown a game over screen.

Another notable puzzle the player may encounter is the jigsaw puzzle which occurs in level 3.Here is player is presented with a 3x4 grid jigsaw puzzle that once solved reveals a portrait of Portia.This puzzle takes an 800x600 picture of portia that is cut into 12 equal pieces each of 200x200 resolution and stored in a folder 'card' within the game's directory and stacks them randomly across a grid.This puzzle is implemented using the python library 'pygame'.

The logic of the game can be explained as follows: First an object of the class puzzle is created.After creation of the object ,the constants that decide where the game is located are set.A table is created followed by the stock ,which is then shuffled so that cards appear in a random order in the grid. The game then distinguishes between the two mouse events-click and drag ,to call corresponding functions to handle these events.

The game also consists of other puzzles such as the shortest path puzzle encountered in level 1 and level 4 in which the player is asked to compute the shortest path between two different nodes in a graph,and a Caesar cipher puzzle encountered in level 5 where the player must decipher a code in order to receive the good ending in the game.In these puzzles the players response is taken as a string using renpy.input() function and then compared with the answer string to check whether it is right in order to proceed forward in the story.

Chapter 7

Testing

1. **Unit Testing**

   - **Objective:** Testing individual units or components in isolation.
   - **Results:** Unit testing was successful, and we identified and fixed several minor bugs within the code. All individual units were functioning as expected.

2. **Integration Testing**

   - **Objective:** Testing the interaction between different units or components.
   - **Results:** Integration testing revealed a few issues with data communication between certain scenes. After addressing these issues, the game demonstrated smooth transitions between scenes.

3. **System Testing**

   - **Objective:** Verifying the entire system as a whole to ensure it meets the specified requirements.
   - **Results:** System testing was comprehensive and revealed a few issues with non-linear story progression. After adjusting the branching narrative, the game played seamlessly across all paths.

4. **Acceptance Testing**

   - **Objective:** Conducted to ensure the game meets end-users' expectations and requirements.
   - **Results:** During acceptance testing, the game was well-received by a group of test players. Feedback was positive, with players praising the engaging narrative and interactive elements.

5. **Cross Device Testing**

   - **Objective:** Testing the game on different devices and platforms to ensure compatibility.
   - **Results:** The game was tested on Windows, macOS, and Android platforms. Minor adjustments were needed for the Android version to optimize performance, but overall, the game ran smoothly on all tested devices.

6. **Security Testing**

   - **Objective:** Identifying and fixing security vulnerabilities that could potentially compromise the game or user data.
   - **Results:** Security testing revealed no critical issues. However, we implemented additional measures to protect user data and prevent unauthorized access.

7. **User Interface Testing**

   - **Objective:** Ensuring that the game's interface is intuitive, user-friendly, and visually appealing.
   - **Results:** User interface testing helped us refine the design and layout of menus, dialogue boxes, and interactive elements. The updated interface was well-received by players.

Chapter 8

Results

Figure 8.1: Main Menu interface

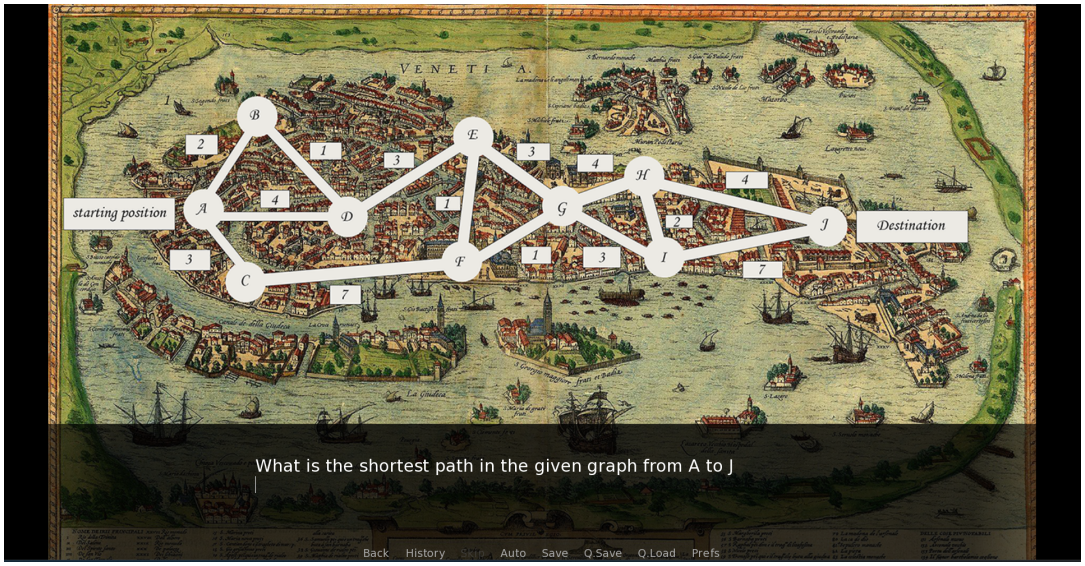

Figure 8.2: Shortest path mini game

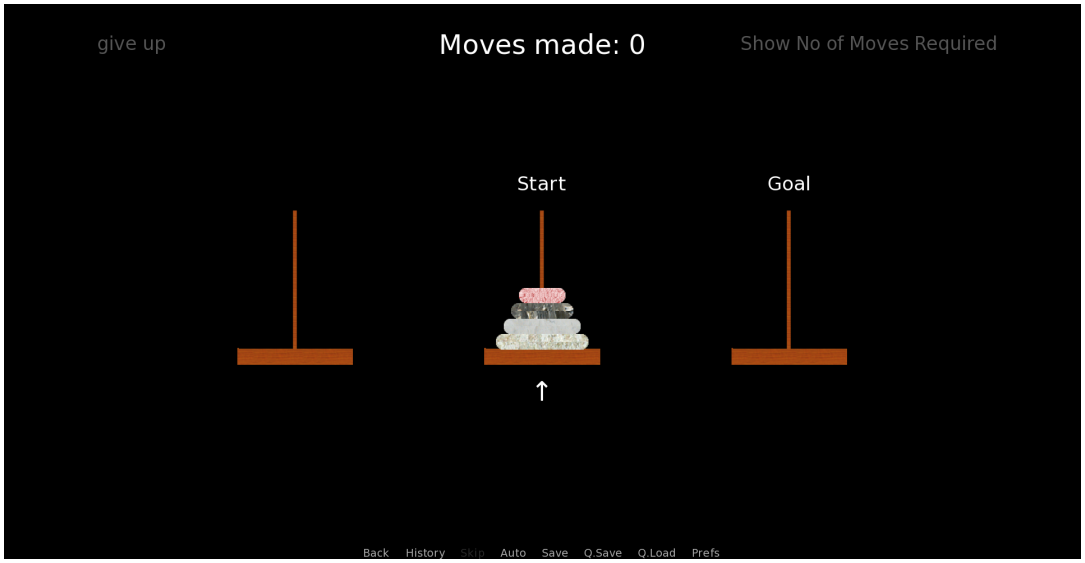

Figure 8.3: Tower of Hanoi Puzzle

Figure 8.4: An example of a Branching choice in the game

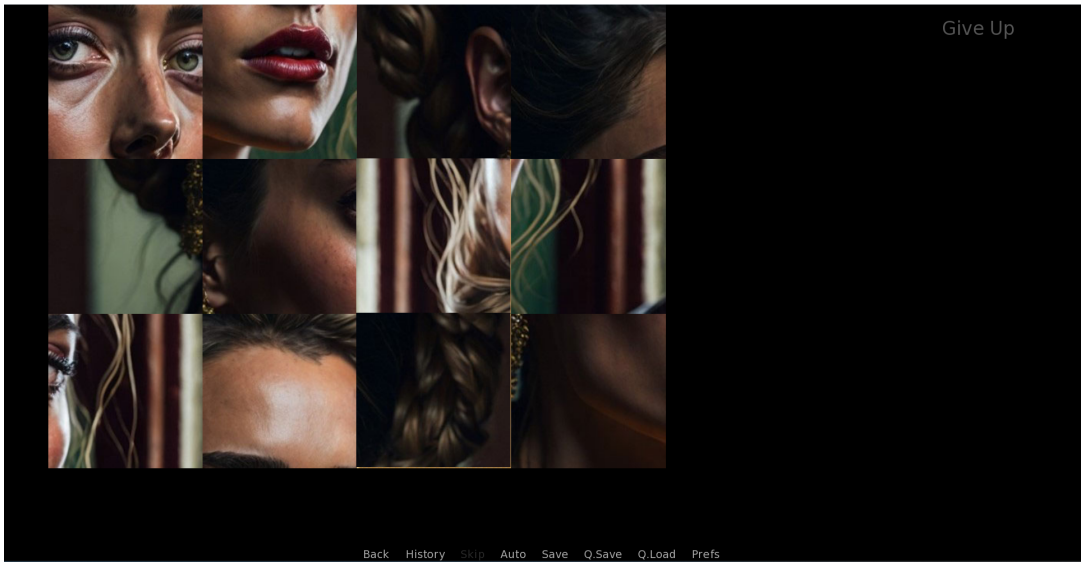

Figure 8.5: Jigsaw puzzle

Figure 8.6: Selectable caskets

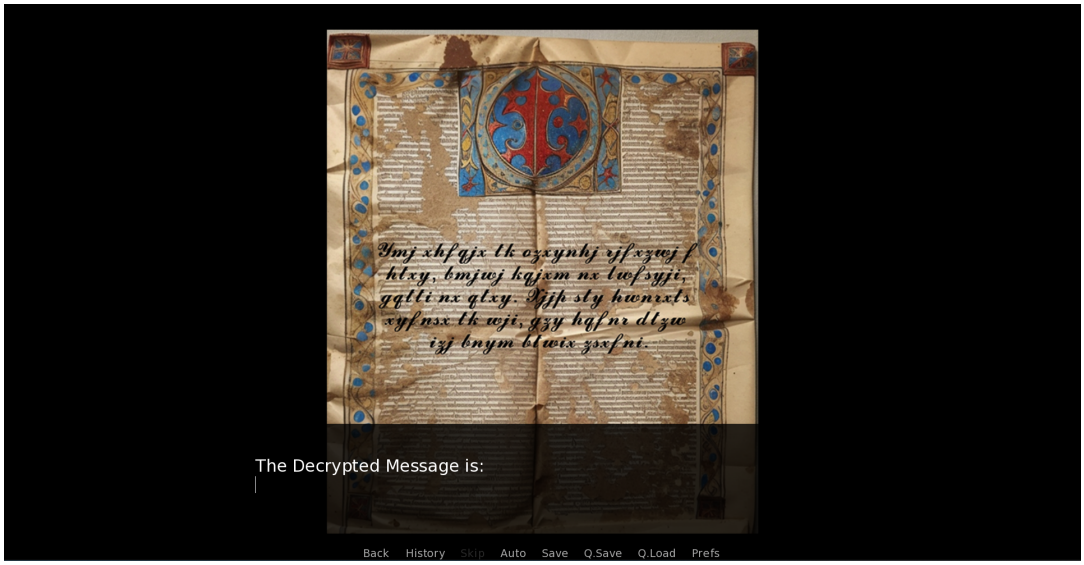

Figure 8.7: Caesar Cipher puzzle

Chapter 9

Risks and Challenges

1. Art and Assets: Creating or acquiring suitable art, character sprites, backgrounds, music, and sound effects can be time-consuming and costly. Ensuring consistent art styles and quality throughout the game can also be a challenge.

2. Storytelling and Writing: Crafting a compelling and engaging storyline that captivates players can be challenging. Balancing character development, plot progression, and player choices requires careful planning and iteration.

3. Game Design and Mechanics: Designing interesting gameplay mechanics within the limitations of a visual novel format can be tricky. Balancing player choices and consequences is essential for an enjoyable experience.

4. Localization and Translation:The game currently only supports english text and hence a player who doesn't know english may find it difficult to understand the story.

5. Intellectual Property and Copyright: Ensuring that all the assets used in the game are properly licensed and not infringing on any copyrights is crucial to avoid legal issues.

Chapter 10

Conclusion

We have developed a visual novel game that brings the timeless tale of William Shakespeare's 'Merchant of Venice' to life through interactive images, engaging puzzles, and dynamic branching choices. The game not only preserves the essence of the classic story but also offers players a unique and immersive experience, blending literature with interactive gameplay.The positive outcomes of the project demonstrate the potential of using Ren'Py as a versatile game engine for visual novel adaptations of literary works.

As the project moves forward, several exciting future scopes can be explored to further enhance and expand the game such as expanding the story by adding more levels and puzzles ,adding multiple language support,creating educational versions that can offer students a fresh approach to understanding classical literature,adding text to speech modules to make it as if the character is speaking ,etc.

References

[1] J. Lebowitz and C. Klug, Interactive Storytelling in Video Games: A Player-Centered Approach to Creating Memorable Characters and Stories, Focal Press, pp.13-268, 2011

[2] D. Cavallaro, Anime and the Visual Novel: Narrative Structure Design and Play at the Crossroads of Animation and Computer Games, McFarland, pp.33-78, 2009.

[3] E. Burgess, Understanding Interactive Fictions as a Continuum: Reciprocity in Experimental Writing Hypertest Fiction and Video Games, The University of Manchester, pp.21-64 ,2015.

[4] Nina Imlig-lten and Dominik Petko,Learning with serious games: Is fun playing the game a predictor of learning success?, pp.2-12, 2016

[5] T. Dodge, The Effects of Interactivity and Visual Realism on Children's Cognitive Empathy Toward Narrative Characters, Indiana University, pp.27-366, 2009.

[6] Y. Chunsoft, T. Yoshimine and Takahiro Ikawa, The Portopia Serial Murder Case ., Enix & Square Enix Holdings Co., Ltd., 1983

[7] Patrick John Harrington Sichter,Embodied Narratives in Video Games: The Stories We Write as We Play, pp.9-55, 2016

[8] Annetta et al ,Investigating the impact of video games on high school students' engagement and learning about genetics, pp.113-167, 2009

[9] Garneli et al ,Computing Education in K-12 Schools: A Review of the Literature, pp.536-540, 2017

[10] Hugh O'Donnell ,Games-Based Learning as an Interdisciplinary Approach to Literacy across Curriculum for Excellence, pp.2-21, 2015

Appendix A: Sample Code

Tower of Hanoi Game Logic

```
image pole = "pole.png"

define texture1 = "texture1.png"

define texture2 = "texture2.png"

define texture3 = "texture3.png"

define texture4 = "texture4.png"


screen hanoi_game_screen(towers):

    textbutton "Show No of Moves Required" action Show("moves_count_scr") align (0.95, 0.05)


    vbox:

        align (0.05, 0.05)

        #textbutton "autosolve" action [ToggleVariable("autosolve", False, True),
SetVariable("create_movements_list", True), Return("start_autosolving")]

        textbutton "give up" action Jump("give_up")


    text "Moves made: [player_moves]" size 35 align (0.5, 0.05)


    for i, each_tower in enumerate(towers):


        ####

        # pole and mark

        vbox:

            pos each_tower["tower_pos"] anchor(0.5, 1.0)

            xalign 0.5


            if each_tower["mark"] == "start":

                text "Start" size 25 xalign 0.5

                null height (block_size*1)


            elif each_tower["mark"] == "finish":

                text "Goal" size 25 xalign 0.5

                null height (block_size*1)


            else:

                text "" size 25 xalign 0.5
```

```
            null height (block_size*1)


        add "pole"


    ####
    # blocks
    vbox:
        pos each_tower["tower_pos"] anchor(0.5, 1.0)
        xalign 0.5 yoffset -20


        for each_block in each_tower["blocks"]:
            frame:
                xpadding 0 ypadding 0
                xmargin 0 ymargin 0
                background Frame(each_block["color"], 10, 10) # color
                xminimum (block_size*(each_block["size"]+2)) xmaximum (block_size*(each_block["size"]+2)) # the
width of block
                yminimum block_size ymaximum block_size
                xalign 0.5


    ####
    # buttons to operate the game
    if can_click:
        ####
        # if player haven't decided yet what block to move
        # and there is some blocks on the pole
        if start_from_tower < 0 and len(each_tower["blocks"])>0:
            button:
                xminimum 125 xmaximum 125
                pos each_tower["tower_pos"] anchor(0.5, 0.0) yoffset 10
                text "↑" size 35 xalign 0.5
                action SetVariable("start_from_tower", i)


        ####
        # if player decided to move block from this pole
        elif start_from_tower == i:
            button:
```

```
                xminimum 125 xmaximum 125
                pos each_tower["tower_pos"] anchor(0.5, 0.0) yoffset 10
                text "undo" size 35 xalign 0.5
                action SetVariable("start_from_tower", -1)

        ####
        # if block that player decided to move is smaller
        # than top block on this pole
        elif (start_from_tower >= 0) and (towers[i]["blocks"] == [] or
towers[start_from_tower]["blocks"][0]["size"]<towers[i]["blocks"][0]["size"]):
            button:
                xminimum 125 xmaximum 125
                pos each_tower["tower_pos"] anchor(0.5, 0.0) yoffset 10
                text "[i] ↓" size 35 xalign 0.5
                action [SetVariable("finish_to_tower", i), Return("move_done")]

        ####
        # if block can't be moved on this pole
        else:
            button:
                xminimum 125 xmaximum 125
                pos each_tower["tower_pos"] anchor(0.5, 0.0) yoffset 10
                text " " size 35 xalign 0.5
                action [[]]

    ####
    # if interactions disabled
    else:
        button:
            xminimum 125 xmaximum 125
            pos each_tower["tower_pos"] anchor(0.5, 0.0) yoffset 10
            text " " size 35 xalign 0.5
            action [[]]


screen moves_count_scr():
    modal True
```

```
    default i = 0
    $ n = hanoi_moves_count(i)

    frame:
        align (0.5,0.5)
        vbox:
            text"Minimal number of moves to solve [i] blocks tower:"
            text "[n]" size 35 xalign 0.5
            null height 20
            hbox:
                xalign 0.5
                textbutton "<" action SetScreenVariable("i", max(0, i-1))
                null width 20
                textbutton ">" action SetScreenVariable("i", i+1)
            null height 20
            textbutton "close" action Hide("moves_count_scr") xalign 0.5



init python:
    def check_game_state():

        global towers, finish_tower, blocks_number
        if len(towers[finish_tower]["blocks"]) == blocks_number:
            return True
        return False


    def hanoi_moves_count(x):

        if x == 1:
            return 1
        elif x > 1:
            return 1 + hanoi_moves_count(x-1) * 2
        else:
            return 0
```

```
def make_path_2(t, x, f):
    # the general list of moves
    global res_2
    res_2 = []

    return hanoi_func_2(t, x, f)

def hanoi_func_2(t, x, f):
    global res_2

    if x in t[0]:
        s = 0
    elif x in t[1]:
        s = 1
    else:
        s = 2

    if (s == f) and (x == 1):
        pass

    elif (s == f) and (x != 1):
        hanoi_func_2(t, x-1, f)

    elif (s != f) and (x == 1):
        res_2.append((s, f))

    else:
        ####
        # ss is the free pole that neither start nor finish one
        # for current tower
        ss = [0,1,2]
        ss.remove(s)
        ss.remove(f)
```

```
        ss = ss[0]
        make_path_1(x-1, ss, f)
        res_2.append((s,  f))
        ####
        # ff is the free pole that neither start nor finish one
        # for current tower
        ff = [0,1,2]
        ff.remove(s)
        ff.remove(f)
        ff = ff[0]
        hanoi_func_2(t, x-1, ff)
    return res_2


def make_path_1(x, s, f):
    global res_1, res_2
    res_1 = []
    hanoi_func_1(x, s, f)

    ####
    # adds all the moves to the general list of moves
    #
    for i in res_1:
        res_2.append(i)


def hanoi_func_1(x, s, f):

    global res_1
    if x == 1:
        res_1.append((s,  f))
    else:
        ####
        # ff is the free pole that neither start nor finish one
        # for current tower
        ff = [0,1,2]
```

```
            ff.remove(s)
            ff.remove(f)
            ff = ff[0]
            hanoi_func_1(x-1, ff, f)
            res_1.append((s,  f))
            hanoi_func_1(x-1, s, ff)
        return res_1

label hanoi_game(blocks_number):  # sets the default number of blocks

    $ block_colors = ["#c00", "#0c0", "#00c", "#cc0", "#c0c", "#0cc", "#930",  "#ccc"]

    $ block_colors = [texture1, texture2, texture3, texture4, texture1, texture2, texture3, texture4]

    if blocks_number > len(block_colors):
        $ blocks_number = len(block_colors)

    ####
    # creates the blocks description
    $ blocks_set = []
    python:
        for b in range(1, blocks_number+1):
            blocks_set.append({"size":  b, "color":block_colors[b-1]} )

    ####
    # randomly sets the start and finish poles
    $ a = [0,1,2]
    $ start_tower = renpy.random.choice(a)
    $ a.remove(start_tower)
    $ finish_tower = renpy.random.choice(a)

    ####
    # the height of the block in pixels
    $ block_size = 20

    ####
```

```
# the positions for all 3 towers
$ towers_pos = [(0.25, 0.65), (0.5, 0.65), (0.75, 0.65)]

####
# main game variable - description of towers
$ towers = [ {"tower_pos":towers_pos[0],"blocks":[], "mark":None},
   {"tower_pos":towers_pos[1],"blocks":[], "mark":None},
   {"tower_pos":towers_pos[2],"blocks":[], "mark":None}
   ]

####
# put the blocks, start and finish marks on their places
$ towers[start_tower]["blocks"] = blocks_set
$ towers[start_tower]["mark"] = "start"
$ towers[finish_tower]["mark"] = "finish"

####
# flags that are used for autosolve function
$ autosolve = False
$ create_movements_list = False

####
# game variables
$ delay_time = 0.2
$ player_moves = 0
$ minimal_moves = hanoi_moves_count(blocks_number)

####
# let's show the game screen
show screen hanoi_game_screen(towers=towers)


####
# main game loop
label hanoi_loop:
```

```
####
# default values that shows that player haven't decided yet
# what move to do
$ start_from_tower = -1
$ finish_to_tower = -1

####
# wait for user interact
if not autosolve:
    $ can_click = True
    $ ui.interact()

####
# to prevent farther interactions
$ can_click = False

####
# autosolve function
if autosolve:

    ####
    # create list of moves
    if create_movements_list:
        ####
        # we need to make this list only once
        $ create_movements_list = False
        $ t = []
        python:
            for i in towers:
                tt = []
                for j in i["blocks"]:
                    tt.append(j["size"])

                t.append(tt)

        ####
```

```
            # and now, let's make the list of moves
            $ movements_list = make_path_2(t, blocks_number, finish_tower)

        ####
        # move to make (the last one in movements_list)
        $ start_from_tower, finish_to_tower = movements_list.pop()


    ####
    # move the block
    $ block_to_move = towers[start_from_tower]["blocks"][0]

    $ towers[start_from_tower]["blocks"] = towers[start_from_tower]["blocks"][1:]
    $ towers[finish_to_tower]["blocks"].insert(0, block_to_move)

    ####
    # add one "move" to counter
    $ player_moves += 1

    ####
    # check the game state
    if check_game_state():
        jump win

    ####
    # make pause between moves if autosolve is enable
    if autosolve:
        $ renpy.pause(delay_time)

    ####
    # next move
    jump hanoi_loop


label win:
    ####
```

```
    # to prevent farther interactions
    $ can_click = False

    if player_moves > minimal_moves:
        "You Lost\nYou took [player_moves] moves.You needed 15 moves"
        hide screen hanoi_game_screen
        jump failed_hanoi
    else:
        "Perfect Win!\nYou took [player_moves] moves."
        hide screen hanoi_game_screen
        scene bg shylock_house_outside
        show antonio_invert at Position(xpos=0.4,xanchor=0.4,ypos=0.1,yanchor=0.1)
        show bassanio at Position(xpos=0.2,xanchor=0.2,ypos=0.1,yanchor=0.1)
        show shylock_guard at Position(xpos=0.9,xanchor=0.9,ypos=0.9,yanchor=0.9)
        shylock_guard "Impressive, you got it right."
        shylock_guard "Hmph, I suppose you have proven yourselves to some extent. Very well, you may enter.
        But remember, the real challenges lie ahead."
        jump see_shylock
    return

label give_up:
    ####
    # to prevent farther interactions
    $ can_click = False

    ####
    # to stop autosolve
    $ autosolve = False

    "You gave up..."

    hide screen hanoi_game_screen

    jump failed_hanoi

# The game starts here.
```

label start_hanoi:

"The guard reveals 3 pegs and 4 disks"

shylock_guard "This is a Tower of Hanoi game.The goal is to move all the disks from one peg to another while following the rules of the puzzle.

If you can solve it in the minimum number of moves, I will grant you access to see Shylock"

scene black

"..."

call hanoi_game(blocks_number=4) from _call_hanoi_game # pass the desirable number of blocks

return

Jigsaw Puzzle Logic

init python:

```
import pygame

DRAG_NONE = 0
DRAG_CARD = 1
DRAG_ABOVE = 2
DRAG_STACK = 3
DRAG_TOP = 4

# Returns the overlap of the area between the two
# rectangles.
def __rect_overlap_area(r1, r2):
    if r1 is None or r2 is None:
        return 0

    x1, y1, w1, h1 = r1
    x2, y2, w2, h2 = r2

    maxleft = max(x1, x2)
    minright = min(x1 + w1, x2 + w2)
    maxtop = max(y1, y2)
    minbottom = min(y1 + h1, y2 + h2)
```

```
        if minright < maxleft:
            return 0

        if minbottom < maxtop:
            return 0

        return (minright - maxleft) * (minbottom - maxtop)

    def __default_can_drag(table, stack, card):
        return table.get_faceup(card)

    class Table(renpy.Displayable):

        def __init__(self, back=None, base=None, springback=0.1, rotate=0.1, can_drag=__default_can_drag,
doubleclick=.33, **kwargs):

            renpy.Displayable.__init__(self, **kwargs)

            # A map from card value to the card object corresponding to
            # that value.
            self.cards = { }

            # A list of the stacks that have been defined.
            self.stacks = [ ]

            # The back of cards that don't have a more specific back
            # defined.
            self.back = renpy.easy.displayable_or_none(back)

            # The base of stacks that don't have a more specific base
            # defined.
            self.base = renpy.easy.displayable_or_none(base)

            # The amount of time it takes for cards to springback
            # into their rightful place.
            self.springback = springback
```

```
# The amount of time it takes for cards to rotate into
# their proper orientation.
self.rotate = rotate

# A function that is called to tell if we can drag a
# particular card.
self.can_drag = can_drag

# The time between clicks for the click to be considered a
# double-click.
self.doubleclick = doubleclick

# Are we sensitive to input? [doc]
self.sensitive = True

# The last click event.
self.last_event = CardEvent()

# The card that has been clicked.
self.click_card = None

# The stack that has been clicked.
self.click_stack = None

# The list of cards that are being dragged.
self.drag_cards = [ ]

# Are we dragging the cards?
self.dragging = False

# The position where we clicked.
self.click_x = 0
self.click_y = 0

# The amount of time we've been shown for.
```

```
    self.st = 0

# This shows the table on the given layer.
def show(self, layer='master'):

    for v in self.cards.itervalues():
        v.offset = __Fixed(0, 0)

    ui.layer(layer)
    ui.add(self)
    ui.close()

# This hides the table.
def hide(self, layer='master'):
    ui.layer(layer)
    ui.remove(self)
    ui.close()

# This controls sensitivity.
def set_sensitive(self,  value):
    self.sensitive = value

def get_card(self, value):
    if value not in self.cards:
        raise Exception("No card has the value %r." % value)

    return self.cards[value]

# This sets the faceup status of a card.
def set_faceup(self, card, faceup=True):
    self.get_card(card).faceup = faceup
    renpy.redraw(self, 0)

def get_faceup(self, card):
    return self.get_card(card).faceup
```

```
# This sets the rotation of a card.
def set_rotate(self, card, rotation):
    __Rotate(self.get_card(card), rotation)
    renpy.redraw(self, 0)

def get_rotate(self, card):
    return self.get_card(card).rotate.rotate_limit()

def add_marker(self, card, marker):
     self.get_card(card).markers.append(marker)
     renpy.redraw(self, 0)

def remove_marker(self, card, marker):
    self.get_card(card).markers.remove(marker)
    renpy.redraw(self, 0)

# Called to create a new card.
def card(self, value, face, back=None):
    self.cards[value] = __Card(self, value, face, back)

# Called to create a new stack.
def stack(self, x, y, xoff=0, yoff=0, show=1024, base=None,
          click=False, drag=DRAG_NONE, drop=False, hidden=False):

    rv = __Stack(self, x, y, xoff, yoff, show, base, click, drag, drop, hidden)

    self.stacks.append(rv)
    return rv

# Force a redraw on each interaction.
def per_interact(self):
    renpy.redraw(self, 0)



def render(self, width, height, st, at):
```

```
        self.st = st

        rv = renpy.Render(width, height)

        for s in self.stacks:

            if s.hidden:
                s.rect = None
                for c in s.cards:
                    c.rect = None
                continue

            s.render_to(rv, width, height, st, at)

            for c in s.cards:
                c.render_to(rv, width, height, st, at)

        return rv

    def event(self, ev, x, y, st):

        self.st = st

        if not self.sensitive:
            return

        grabbed = renpy.display.focus.get_grab()

        if (grabbed is not None) and (grabbed is not self):
            return

        if ev.type == pygame.MOUSEBUTTONDOWN and ev.button == 1:

            if self.click_stack:
                return
```

```
stack = None
card = None

for s in self.stacks:

    sx, sy, sw, sh = s.rect
    if sx <= x and sy <= y and sx + sw > x and sy + sh > y:
        stack = s



    for c in s.cards[-s.show:]:
        if c.rect is None:
            continue


        cx, cy, cw, ch = c.rect
        if cx <= x and cy <= y and cx + cw > x and cy + ch > y:
            card = c
            stack = c.stack

if stack is None:
    return

# Grab the display.
renpy.display.focus.set_grab(self)

# Don't let the user grab a moving card.
if card is not None:
    xoffset, yoffset = card.offset.offset()
    if xoffset or yoffset:
        return

# Move the stack containing the card to the front.
self.stacks.remove(stack)
self.stacks.append(stack)

if stack.click or stack.drag:
```

```
            self.click_card = card
            self.click_stack = stack

        if card is None or not self.can_drag(self, card.stack, card.value):
            self.drag_cards = [ ]
        elif card.stack.drag == DRAG_CARD:
            self.drag_cards = [ card ]
        elif card.stack.drag == DRAG_ABOVE:
            self.drag_cards = [ ]
            for c in card.stack.cards:
                if c is card or self.drag_cards:
                    self.drag_cards.append(c)
        elif card.stack.drag == DRAG_STACK:
            self.drag_cards = list(card.stack.cards)
        elif card.stack.drag == DRAG_TOP:
            if card.stack.cards[-1] is card:
                self.drag_cards = [ card ]
            else:
                self.drag_cards = [ ]

        for c in self.drag_cards:
            c.offset = __Fixed(0, 0)

        self.click_x = x
        self.click_y = y
        self.dragging = False

        renpy.redraw(self, 0)

        raise renpy.IgnoreEvent()

    if ev.type == pygame.MOUSEMOTION or (ev.type == pygame.MOUSEBUTTONUP and ev.button == 1):

        if abs(x - self.click_x) > 7 or abs(y - self.click_y) > 7:
            self.dragging = True
```

```
dx = x - self.click_x
dy = y - self.click_y

for c in self.drag_cards:
    xoffset, yoffset = c.offset.offset()

    cdx = dx - xoffset
    cdy = dy - yoffset

    c.offset = __Fixed(dx, dy)

    if c.rect:
        cx, cy, cw, ch = c.rect
        cx += cdx
        cy += cdy
        c.rect = (cx, cy, cw, ch)

area = 0
dststack = None
dstcard = None

for s in self.stacks:
    if not s.drop:
        continue

    for c in self.drag_cards:

        if c.stack == s:
            continue
        a = __rect_overlap_area(c.rect,  s.rect)
        if a >= area:
            dststack = s
            dstcard = None
            area = a

        for c1 in s.cards:
```

```
                a = __rect_overlap_area(c.rect,  c1.rect)
                if a >= area:
                    dststack = s
                    dstcard = c1
                    area = a

        if area == 0:
            dststack = None
            dstcard = None

        renpy.redraw(self, 0)

        if ev.type == pygame.MOUSEMOTION:
            raise renpy.IgnoreEvent()

    if ev.type == pygame.MOUSEBUTTONUP and ev.button == 1:

        # Ungrab the display.
        renpy.display.focus.set_grab(None)

        evt = None

        if self.dragging:
            if dststack is not None and self.drag_cards:

                evt = CardEvent()
                evt.type = "drag"
                evt.table = self
                evt.stack = self.click_stack
                evt.card = self.click_card.value
                evt.drag_cards = [c.value for c in self.drag_cards]
                evt.drop_stack = dststack
                if dstcard:
                    evt.drop_card = dstcard.value
                evt.time = st
```

```
else:

    if self.click_stack.click:

        evt = CardEvent()
        evt.type = "click"
        evt.table = self
        evt.stack = self.click_stack
        if self.click_card:
            evt.card = self.click_card.value
        else:
            evt.card = None

        evt.time = st

        if (evt.type == self.last_event.type
            and evt.stack == self.last_event.stack
            and evt.card == self.last_event.card
            and evt.time < self.last_event.time  + self.doubleclick):

            evt.type = "doubleclick"

if evt is not None:
    self.last_event = evt

for c in self.drag_cards:
    c.springback()

self.click_card = None
self.click_stack = None
self.drag_cards = [ ]

if evt is not None:
    return evt
else:
    raise renpy.IgnoreEvent()
```

```
class CardEvent(object):

    def __init__(self):
        self.type = None
        self.stack = None
        self.card = None
        self.drag_cards = None
        self.drop_stack = None
        self.drop_card = None
        self.time = 0

# Represents a stack of one or more cards, which can be placed on the
# table.
class __Stack(object):

    def __init__(
        self, table,
        x, y,
        xoff, yoff,
        show, base,
        click, drag, drop,
        hidden):


        # The table this stack belongs to.
        self.table = table

        # The x and y coordinates of the center of the top card of
        # this stack.
        self.x = x
        self.y = y

        # The offset in the x and y directions of each successive
        # card.
```

```
        self.xoff = xoff
        self.yoff = yoff

        # The number of cards to show from this stack. (We show the
        # last show cards if this is less than the numebr of cards
        # in the stack.)
        self.show = show

        # The image that is shown behind the stack. If None, the
        # background is taken from the table.
        self.base = base

        # Should we report click events on this stack?
        self.click = click

        # Should we allow dragging this stack? If so, how?
        self.drag = drag

        # Should we allow dropping to this stack?
        self.drop = drop

        # Is this stack hidden?
        self.hidden = hidden

        # The list of cards in this stack.
        self.cards = [ ]

        # The rectangle for the background of this effect.
        self.rect = None

    def insert(self, index, card):
        card = self.table.get_card(card)

        if card.stack:
            card.stack.cards.remove(card)
```

```
        card.stack = self
        self.cards.insert(index, card)

        self.table.stacks.remove(self)
        self.table.stacks.append(self)

        card.springback()

    def append(self, card):
        if card in self.cards:
            self.insert(len(self.cards) - 1, card)
        else:
            self.insert(len(self.cards), card)

    def remove(self, card):
        card = self.table.get_card(card)
        self.cards.remove(card)
        card.stack = None
        card.rect = None

    def deal(self):
        if not self.cards:
            return None

        card = self.cards[-1]
        self.remove(card.value)
        return card.value

    def shuffle(self):
        renpy.random.shuffle(self.cards)
        renpy.redraw(self.table, 0)

    def __len__(self):
        return len(self.cards)

    def __getitem__(self, idx):
```

```
        return self.cards[idx].value

    def __iter__(self):
        for i in self.cards:
            yield i.value

    def __contains__(self, item):
        return self.table.get_card(card) in self.cards

    def render_to(self, rv, width, height, st, at):

        base = self.base or self.table.base

        if base is None:
            return

        surf = renpy.render(base, width, height, st, at)
        cw, ch = surf.get_size()

        cx = self.x - cw / 2
        cy = self.y - ch / 2

        self.rect = (cx, cy, cw, ch)
        rv.blit(surf, (cx, cy))

class __Card(object):

    def __init__(self, table, value, face, back):

        # The table this card belongs to.
        self.table = table

        # The value of this card.
        self.value = value

        # The face of this card.
```

```
        self.face = renpy.easy.displayable(face)

        # The back of this card. If None, then the back is taken from
        # the table the card belongs to.
        self.back = renpy.easy.displayable_or_none(back)

        # Is this card faceup (or face down?)
        self.faceup = True

        # Object that's called to decide how rotated this card should
        # be.
        self.rotate = None

        # A series of highlights that should be drawn over this card.
        self.highlights = [ ]

        # The stack this card is in.
        self.stack = None

        # An object that gives the offset of this card relative to
        # where it would normally be placed.
        self.offset = __Fixed(0, 0)

        # The rectangle where this card was last drawn to the screen
        # at.
        self.rect = None

        __Rotate(self, 0)

    # Returns the base x and y placement of this card.
    def place(self):
        s = self.stack
        offset = max(len(s.cards) - s.show, 0)
        index = max(s.cards.index(self) - offset, 0)

        return (s.x + s.xoff * index, s.y + s.yoff * index)
```

```
def springback(self):
    if self.rect is None:
        self.offset = __Fixed(0, 0)
    else:
        self.offset = __Springback(self)

def render_to(self, rv, width, height, st, at):

    x, y = self.place()
    xoffset, yoffset = self.offset.offset()
    x += xoffset
    y += yoffset

    if self.faceup:
        d = self.face
    else:
        d = self.back or self.table.back

    # TODO: Figure out if we can reuse some of this.

    if self.highlights:
        d = Fixed(* ([d] + [renpy.easy.displayable(i) for i in self.highlights]))

    r = self.rotate.rotate()
    if r:
        d = RotoZoom(r, r, 0, 1, 1, 0)(d)

    surf = renpy.render(d, width, height, st, at)
    w, h = surf.get_size()

    x -= w / 2
    y -= h / 2

    self.rect = (x, y, w, h)
```

```
        rv.blit(surf, (x, y))

    def __repr__(self):
        return "<__Card %r>" % self.value


class __Springback(object):

    def __init__(self, card):
        self.card = card
        self.table = table = card.table

        self.start = table.st

        cx, cy, cw, ch = self.card.rect
        x = cx + cw / 2
        y = cy + ch / 2

        self.startx = x
        self.starty = y

    def offset(self):

        t = (self.table.st - self.start) / self.table.springback
        t = min(t, 1.0)

        if t < 1.0:
            renpy.redraw(self.table, 0)

        px, py = self.card.place()

        return int((self.startx - px) * (1.0 - t)), int((self.starty - py) * (1.0 - t))


class __Fixed(object):
    def __init__(self, x, y):
```

```
        self.x = x
        self.y = y

    def offset(self):
        return self.x, self.y

class __Rotate(object):
    def __init__(self, card, amount):

        self.table = table = card.table
        self.start = table.st

        if card.rotate is None:
            self.start_rotate = amount
        else:
            self.start_rotate = card.rotate.rotate()

        self.end_rotate = amount

        card.rotate = self

    def rotate(self):

        if self.start_rotate == self.end_rotate:
            return self.start_rotate

        t = (self.table.st - self.start) / self.table.springback
        t = min(t, 1.0)

        if t < 1.0:
            renpy.redraw(self.table, 0)

        return self.start_rotate + (self.end_rotate - self.start_rotate) * t

    def rotate_limit(self):
        return self.end_rotate
```

Appendix B: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	DESCRIPTION	Blooms' Taxonomy Level
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS622T.1-PO1	**HIGH**	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.1-PO2	**HIGH**	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS622T.1-PO3	**HIGH**	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS622T.1-PO4	**HIGH**	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS622T.1-PO6	**MEDIUM**	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS622T.1-PO7	**MEDIUM**	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS622T.1-PO8	**HIGH**	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS622T.1-PO9	**MEDIUM**	Identify technically and economically feasible problems by working as a team.
100003/CS622T.1-PO10	**MEDIUM**	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS622T.1-P011	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS622T.1-PO12	**HIGH**	Identify technically and economically feasible problems for long term learning.
100003/CS622T.1-PSO1	**MEDIUM**	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS622T.1-PSO2	**MEDIUM**	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS622T.1-PSO3	**MEDIUM**	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS622T.2-PO1	**HIGH**	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS622T.2-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS622T.2-PO3	**HIGH**	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS622T.2-PO4	**HIGH**	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS622T.2-PO5	**HIGH**	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS622T.2-PO6	**MEDIUM**	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS622T.2-PO8	**HIGH**	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS622T.2-PO9	**MEDIUM**	Identify and survey the relevant literature as a team.
100003/CS622T.2-PO10	**HIGH**	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS622T.2-PO11	**MEDIUM**	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS622T.2-PO12	**HIGH**	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS622T.2-PSO1	**MEDIUM**	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS622T.2-PSO2	**MEDIUM**	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS622T.2-PSO3	**MEDIUM**	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS622T.3-PO1	**HIGH**	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS622T.3-PO2	**HIGH**	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS622T.3-PO3	**HIGH**	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO4	**HIGH**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.3-PO5	**HIGH**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS622T.3-PO6	**MEDIUM**	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS622T.3-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PO8	**HIGH**	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS622T.3-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.3-PO10	**MEDIUM**	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS622T.3-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS622T.4-PO1	**MEDIUM**	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.4-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS622T.4-PO3	**MEDIUM**	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS622T.4-PO4	**MEDIUM**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS622T.4-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS622T.4-PO8	**HIGH**	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS622T.4-PO9	**HIGH**	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.4-PO10	**HIGH**	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS622T.4-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS622T.4-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS622T.4-PSO1	**MEDIUM**	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS622T.4-PSO2	**MEDIUM**	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS622T.4-PSO3	**MEDIUM**	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS622T.5-PO1	**HIGH**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.5-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PO3	**HIGH**	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS622T.5-PO4	**MEDIUM**	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.5-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO6	**MEDIUM**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO8	**HIGH**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO1	**MEDIUM**	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PSO2	**MEDIUM**	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.