**Wandering in Woods**

**Summer – II**

**Design Document**

**August 25, 2023**

**Table of Contents**

[**Introduction** 3](#_30j0zll)

[**Purpose** 3](#_1fob9te)

[**Wandering in the Woods** 3](#_3znysh7)

[**Process Model** 4](#_2et92p0)

[**Use Case** 5](#_tyjcwt)

[**Use Case 1: Launch Game and Start Playing** 6](#_3dy6vkm)

[**Use Case 2: Begin Play** 6](#_1t3h5sf)

[**Use Case 3: Play Game and Make Moves** 7](#_4d34og8)

[**Use Case 4: Choose Stage** 7](#_2s8eyo1)

[**Use Case 5: View Game Statistics** 8](#_17dp8vu)

[**Use Case 6: Replay Game** 8](#_3rdcrjn)

[**Use Case 7: Exit Game** 9](#_26in1rg)

[**Use Case 8: Move Diagonal** 9](#_lnxbz9)

[**Use Case 9: Move Randomly and Place Characters** 9](#_35nkun2)

[**Use Case 10: Change Grid Size** 10](#_1ksv4uv)

[**Use Case 11: Play Challenges** 10](#_44sinio)

[**Use Case 12: Test Wandering Methods** 11](#_2jxsxqh)

[**UML Model** 11](#_z337ya)

[**Use Case Diagram** 11](#_3j2qqm3)

[**Deployment Diagram** 13](#_1y810tw)

[**Class Diagram** 13](#_4i7ojhp)

[**State Diagram** 17](#_2xcytpi)

[**Activity Diagram** 18](#_1ci93xb)

[**Customer Journey Map** 19](#_3whwml4)

[**Personas** 21](#_2bn6wsx)

# **Introduction**

## **Purpose**

This detailed plan outlines the Wandering in the Woods game development process. It outlines game development procedures and considerations. Agile development provides for flexibility and modification as the project evolves. This agile approach accepts that game design may change, unlike traditional planning.

The first iteration of development defines the System Architecture model. This model specifies how game pieces will function together. New knowledge and data will enhance and expand design models. These models may not be comprehensive, but they will include the game's essential subsystems and components.

According to Agile principles, this document only contains information the design team considers vital for preservation and regular updates. This guarantees the document remains dynamic and relevant during development. The developers view this paper as a draft of the final plan. Critical feedback can enhance the paper and game structure.

This document is also useful for students studying computer simulation concepts. This game prototype instructs elementary school students about computing, arithmetic, and programming. Young learners may study these things interactively and easily by playing the game.

Additionally, this document provides a complete summary of Wandering in the Woods. It explores the game's main objectives and features, explaining its design. It also describes the game's numerous interactions. This document serves stakeholders and designers involved in the game's development.

Overall, this document provides a complete tutorial for making Wandering in the Woods. It emphasizes iterative design and adaptability like agile. This keeps the paper relevant as a dynamic blueprint. This template helps the design of an instructive and fun game that teaches essential information.

## **Wandering in the Woods**

For elementary school children, "Wandering in the Woods" teaches basic concepts in a pleasant and engaging way. The game targets grades three through five and six through eight children's cognitive abilities and learning goals.

The game's graphic interface is basic but useful for grades three to five. It introduces "little data," which arises from player activities in the game. This interaction presents students with decision-making scenarios that require data analysis and problem-solving. The game promotes critical thinking and teaches causality, showing students how their actions affect results.

For grades six through eight, the game includes an advanced edition that explores data processing and analysis. Here, students see huge and small datasets to grasp data scalability. They learn data representation through creating charts and graphs from data. The game's actions have more implications, teaching them how decision-making affects data results.

The game's simulation is designed for two students to work together on a computer screen to optimize learning. As students strategize and evaluate game data, this collaborative component promotes collaboration.

Auditory cues and instructions help players understand the game. This multi-modal approach lets students with different learning styles enjoy the game. The game also tests pupils' work completion. These tests help educators and parents customize assistance and advice by revealing each student's comprehension and development.

# **Process Model**

Process models provide a structured framework for businesses to analyze, manage, and improve internal business processes. This streamlines procedures and boosts efficiency. This technique generally involves continual improvements that follow agile’s flexibility and adaption principles.

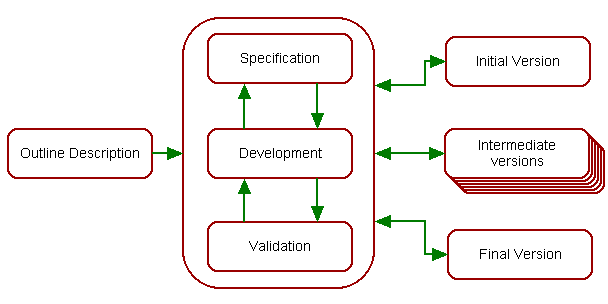
The game's development will use the evolutionary software development life cycle evolution paradigm. This methodology combines iterative and incremental game design to create a dynamic framework. The evolutionary development model (EVO) aims to build products that meet audience preferences while reducing costs and risks. The methodology also allows fast product improvements, supporting rapid advancement.

The evolutionary method encourages projects to be broken down strategically. These components are given in phases dependent on client importance. This modular method facilitates step-by-step development. The evolutionary technique guarantees that the customer receives concrete outcomes throughout the project, building trust and confidence.

This strategy delivers several manageable chunks to the consumer. These incremental delivery strategies allow for regular input and keep the project in line with the client's changing demands. The development team strengthens client relationships by regularly delivering measurable goods and services.

Importantly, the evolutionary process model adapts to project needs. This flexibility recognizes changing consumer demands and software development. The paradigm also makes all work portions manageable, facilitating updates and additions.

The evolutionary process paradigm guides game development strategy. The strategy uses iterative development and incremental delivery to build a game that fulfills client expectations and adjusts to project and audience needs. This methodology improves game development efficiency, transparency, and effectiveness through modularity, frequent releases, and adaptation.



*Figure 1: Evolutionary development process model for system.*

# **Use Case**

In a variety of contexts, people take diverse activities inside a system. These are user-perceived activities. User goals are the beginning point and end goal of each scenario. These scenarios explain how the system should work and help predict issues. A list of objectives evaluates a system's cost and complexity. After identifying essential functions, the project team may choose which ones to build. Use case design should examine what is inside and outside the process. Other use cases or supporting actors identify essential components outside a use case's scope. Design might focus on systems, subsystems, or organization. Businesses define operational procedures with use cases.

After analyzing a scenario, the design team created the following examples, as described below:

## **Use Case 1: Launch Game and Start Playing**

| Use Case ID: | 1 |
| --- | --- |
| Description: | The student launches the game, navigates to the main page, and starts the game by selecting the "Start Game" option. |
| Actor: | Student |
| Pre-Condition: | The game must be installed on the system. |
| Post-Condition: | The game is loaded, and the student starts playing. |
| Flow of Events: | 1. The student clicks the game icon. 2. The student selects "Start Game". |
| Notes | The main page might include game instructions or a welcome message. |

## **Use Case 2: Begin Play**

| Use Case ID: | 2 |
| --- | --- |
| Description: | The student selects a specific stage, starts playing the game, and navigates to the corresponding grid page. |
| Actor: | Student |
| Pre-Condition: | Student must choose a stage from available options. |
| Post-Condition: | The game starts with the selected stage's grid. |
| Flow of Events: | 1. The student starts the game. 2. The student selects "Choose Stage". 3. The student presses "Play". |
| Notes | Different stages might have unique challenges or scenarios. |

## **Use Case 3: Play Game and Make Moves**

| Use Case ID: | 3 |
| --- | --- |
| Description: | The student plays the game, navigates to the grid page, and makes moves within the game. |
| Actor: | Student |
| Pre-Condition: | The game must be in play mode. |
| Post-Condition: | The student interacts with the grid and progresses in the game. |
| Flow of Events: | 1. The student starts the game. 2. The student presses "Play". 3. The system displays the grid. 4. The student makes moves within the grid. |
| Notes | The grid might include various obstacles or objectives. |

## **Use Case 4: Choose Stage**

| Use Case ID: | 4 |
| --- | --- |
| Description: | The student selects a stage from available options, starts playing the game, and navigates to the corresponding grid page. |
| Actor: | Student |
| Pre-Condition: | The student must choose a stage. |
| Post-Condition: | The system displays the grid page based on the selected stage. |
| Flow of Events: | 1. The student clicks the game icon. 2. The student starts the game. 3. The student selects "Choose Stage". 4. The student presses "Play". 5. The system displays the game screen with the selected stage's grid. 6. The student starts playing. |
| Included Use Case: | Choose Stage |
| Notes | The selected stage determines the level of difficulty and challenges. |

## **Use Case 5: View Game Statistics**

| Use Case ID: | 5 |
| --- | --- |
| Description: | The student views game statistics related to their gameplay. |
| Actor: | Student |
| Pre-Condition: | The student must play the game. |
| Post-Condition: | The system periodically displays gameplay statistics. |
| Flow of Events: | 1. The student starts the game. 2. The student selects "Choose Stage". 3. The student presses "Play". 4. The system displays the grid. 5. The student starts playing. 6. The system displays game statistics. |
| Notes | The statistics might include score, time played, or other relevant metrics. |

## **Use Case 6: Replay Game**

| Use Case ID: | 6 |
| --- | --- |
| Description: | The student replays the game multiple times. |
| Actor: | Student |
| Pre-Condition: | The student must play the game. |
| Post-Condition: | The system replays the game upon the student's request. |
| Flow of Events: | 1. The student starts the game. 2. The student selects "Choose Stage". 3. The student presses "Play". 4. The system displays the grid. 5. The student starts playing. 6. The system displays game statistics. 7. The student requests to replay the game. |
| Notes | The student might want to improve their performance or explore different strategies. |

## **Use Case 7: Exit Game**

| Use Case ID: | 7 |
| --- | --- |
| Description: | The student exits and ends the game. |
| Actor: | Student |
| Pre-Condition: | The student must be inside the game screen. |
| Post-Condition: | The game is closed, and the window is exited. |
| Flow of Events: | 1. The student clicks the game icon. 2. The student starts the game. 3. The student exits the game. |
| Notes | The student might need to leave the game for various reasons. |

## **Use Case 8: Move Diagonal**

| Use Case ID: | 8 |
| --- | --- |
| Description: | The student makes a diagonal move in the game. |
| Actor: | K-2 Group Student |
| Pre-Condition: | The student must be from K-2 group and the game must be started. |
| Post-Condition: | The system counts the move and displays happy graphics if two students meet. |
| Flow of Events: | 1. The student starts the game. 2. The student presses "Play". 3. The system displays the grid. 4. The student makes a diagonal move. 5. If two students meet, the system displays happy graphics. 6. The system resets the game. |
| Notes | The system promotes interaction and engagement between young students. |

## **Use Case 9: Move Randomly and Place Characters**

| Use Case ID: | 9 |
| --- | --- |
| Description: | The student moves characters randomly on the grid and places them at specific locations. |
| Actor: | K3-5 Group Student, K 6-8 Group Student |
| Pre-Condition: | The student must be from K3-5 or K6-8 group, and the game must be started. |
| Post-Condition: | The system counts the moves, stores statistics, and updates the grid. |
| Flow of Events: | 1. The student starts the game. 2. The student presses "Play". 3. The system displays the grid. 4. The student places characters anywhere on the grid. 5. The student makes moves in any direction. 6. If two students meet, the system displays happy graphics. 7. The system resets the game. |
| Notes | Encourages creativity and experimentation among students. |

## **Use Case 10: Change Grid Size**

| Use Case ID: | 10 |
| --- | --- |
| Description: | The student changes the size of the grid before playing the game. |
| Actor: | K3-5 Group Student, K 6-8 Group Student |
| Pre-Condition: | The student must be from K3-5 or K6-8 group, and the game must be started. |
| Post-Condition: | The system counts the moves, stores statistics, and adapts the grid size. |
| Flow of Events: | 1. The student starts the game. 2. The student presses "Play". 3. The system displays the grid. 4. The student changes the size of the grid. 5. The student makes moves in any direction. 6. If two students meet, the system displays happy graphics. 7. The system resets the game. |
| Notes | Allows students to tailor the game experience to their preferences. |

## **Use Case 11: Play Challenges**

| Use Case ID: | 11 |
| --- | --- |
| Description: | The student engages in challenges presented by the system during gameplay. |
| Actor: | K 6-8 Group Student |
| Pre-Condition: | The student must be from the K6-8 group, and the game must be started. |
| Post-Condition: | The system counts the moves, stores statistics, and offers various challenges. |
| Flow of Events: | 1. The student starts the game. 2. The student presses "Play". 3. The system displays the grid. 4. The system presents challenges to the student. 5. The student makes moves in any direction. 6. If two students meet, the system displays happy graphics. 7. The system resets the game. |
| Notes | Enhances problem-solving skills and adds dynamic gameplay. |

## **Use Case 12: Test Wandering Methods**

| Use Case ID: | 12 |
| --- | --- |
| Description: | The student experiments with different wandering methods for a short time. |
| Actor: | K 6-8 Group Student |
| Pre-Condition: | The student must be from the K6-8 group, and the game must be started. |
| Post-Condition: | The system counts the moves, stores statistics, and presents wandering paths. |
| Flow of Events: | 1. The student starts the game. 2. The student presses "Play". 3. The system displays the grid. 4. The system tests different wandering methods. 5. The student makes moves in any direction. 6. If two students meet, the system displays happy graphics. 7. The system resets the game. |
| Notes | Encourages exploration of various strategies and tactics |

# **UML Model**

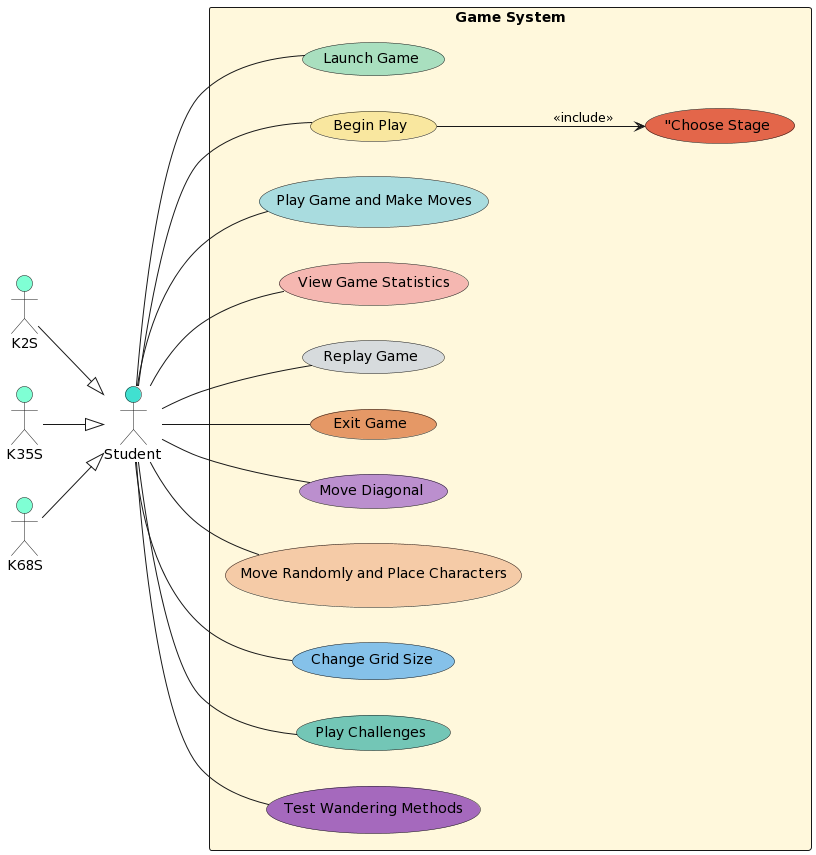
## **Use Case Diagram**

Use case diagrams help define a system's dynamic nature. The study of various systems often uses UML use case diagrams. They are essential for revealing system role interactions. These diagrams help explain role interaction.

Use case diagrams extract system requirements. These diagrams cover internal dynamics and external impacts, making them ideal for determining system requirements, especially design ones. Creating a system's features requires careful use case preparation and actor identification.

A system's main functions and actions are crucial to utilize case diagrams. This visualization helps identify actors and their relationships. A use case diagram describes what a system does and how actors utilize it, but it does not describe its fundamental operations.

As shown in Figure 2, the use case diagram clearly shows a system's usage situations using these concepts. The students from grade 2 are represented using “K2S”. Students from grade 3-5 by “K35S” and students from 6-8Studentsre represented as “K68S”. This representation communicates the system's goal, user interactions, and features, which are essential to understanding its operating dynamics.



*Figure 2: Use case diagram for wandering in woods system.*

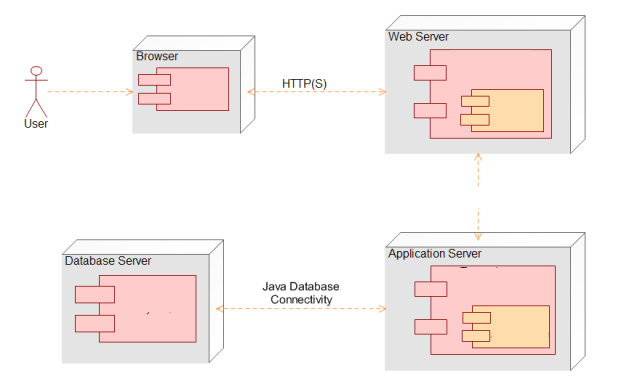
## **Deployment Diagram**

Deployment diagrams show a system's hardware, connectivity, and software layout. UML deployment diagrams show system architecture. These diagrams show how hardware and software interact in a system.

A deployment diagram shows how hardware and software elements interact in a system. The system's infrastructure physically distributes computing workloads and shows their link.

The system's physical blueprint is the deployment diagram in Figure 3. Given the option to develop the system in Java, the graphic shows Java connection integration. The web server, Apache, effectively handles client replies to promote communication. The application server uses servlet-based apps to store data in a SQL database.

This comprehensive Figure shows how hardware and software work together to provide the system's intended functions. It describes the strategic placement of various components, helping to understand the system's deployment architecture.



*Figure 3: Deployment diagram for system.*

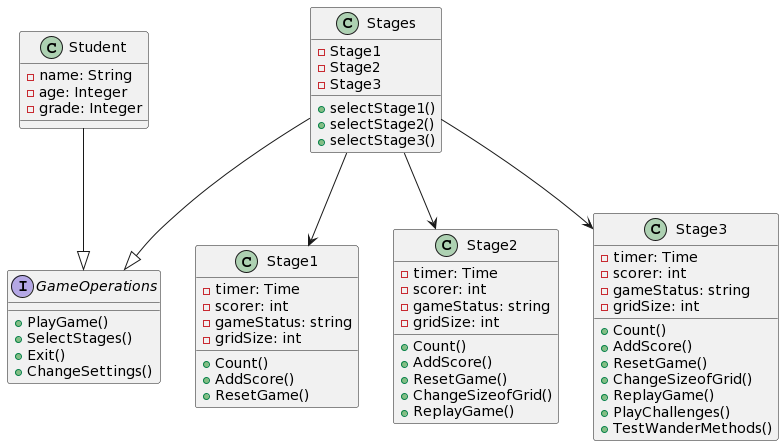
## **Class Diagram**

Class diagrams are essential for displaying a system's static structural elements. They show the organization and interactions between application segments. Class diagrams are a graphical representation, documentation center, and system explication tool, as well as a basis for generating software code.

Class diagrams bring the application's static appearance to life by displaying its architectural classes. Each system class represents a separate notion. These diagrams explain how classes interact and collaborate and provide design justification.

The class diagram in Figure 4 shows the system's structure. This diagram explains system class organization. Given the three game stages, the diagram presents three classes, each for a stage. These classes precisely specify the necessary functions to capture each game stage's distinct behaviors and actions.

These class diagrams reveal how different classes interact and contribute to the system's operation. These diagrams also help software engineers turn conceptual design into code.



*Figure 4: Class diagram for system.*

**Description of classes:**

**Class: Student**

| Attributes | Description |
| --- | --- |
| name | Name of the student (string) |
| age | Age of the student (integer) |
| grade | Grade of the student (integer) |

**Operations**

| Function | Description |
| --- | --- |
| PlayGame() | Play the game |
| SelectStages() | Select stages in the game |
| Exit() | Exit the game |
| ChangeSettings() | Change game settings |

**Class: Stages**

| Attributes | Description |
| --- | --- |
| Stage 1 | Information about stage 1 |
| Stage 2 | Information about stage 2 |
| Stage 3 | Information about stage 3 |

**Operations**

| Function | Description |
| --- | --- |
| SelectStage1() | Select stage 1 of the game |
| SelectStage2() | Select stage 2 of the game |
| SelectStage3() | Select stage 3 of the game |

**Class: Stage 1**

| Attributes | Description |
| --- | --- |
| timer | Time to store each move of the student |
| scorer | Score of each student |
| Gamestatus | Status of the game (play, resume, exit) |
| Gridsize | Size of the grid |

**Operations**

| Function | Description |
| --- | --- |
| Count() | Count the moves of the student |
| AddScore() | Add scores for each student's move |
| ResetGame() | Reset the game to initial state |

**Class: Stage 2**

| Attributes | Description |
| --- | --- |
| timer | Time to store each move of the student |
| scorer | Score of each student |
| Gamestatus | Status of the game (play, resume, exit) |
| Gridsize | Size of the grid |

**Operations**

| Function | Description |
| --- | --- |
| Count() | Count the moves of the student |
| AddScore() | Add scores for each student's move |
| ResetGame() | Reset the game to initial state |
| ChangeSizeofGrid() | Change the size of the grid |
| ReplayGame() | Replay the game |

**Class: Stage 3**

| Attributes | Description |
| --- | --- |
| timer | Time to store each move of the student |
| scorer | Score of each student |
| Gamestatus | Status of the game (play, resume, exit) |
| Gridsize | Size of the grid |

**Operations**

| Function | Description |
| --- | --- |
| Count() | Count the moves of the student |
| AddScore() | Add scores for each student's move |
| ResetGame() | Reset the game to initial state |
| ChangeSizeofGrid() | Change the size of the grid |
| ReplayGame() | Replay the game |
| PlayChallenges() | Present new challenges to students |
| TestWanderMethods() | Try and test different methods |

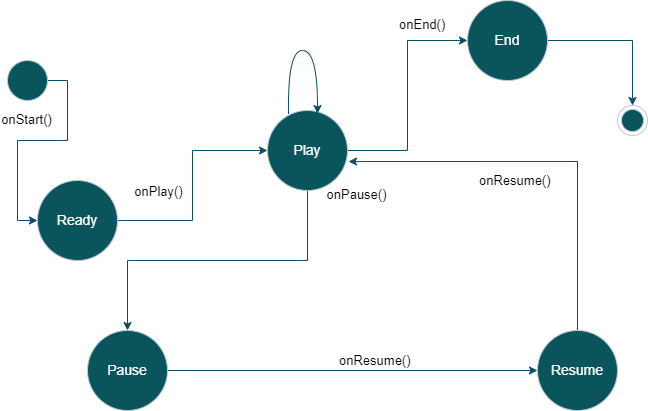
## **State Diagram**

System state diagrams demonstrate how a system or component within it changes over time. This behavioral diagram shows system activity via finite state transitions. The phrases state machines and state chart diagrams are sometimes interchangeable. State diagrams show how a class reacts to time and external inputs. It is crucial for one to understand that state diagrams do not model all classes' states.

The state diagram in Figure 5 shows the system's behavior in different states. This diagram shows how the system switches states based on user inputs and time.

Starting the game puts the system in the "READY" state. Selecting "Play" changes the state to "PLAY," indicating current gameplay. A gaming pause puts the system in the "PAUSE" state. The system returns to the "RESUME" state after resuming gaming. Finally, leaving the game puts the state in the "END" state, ending the gaming experience.

This dynamic model shows how user activities and temporal changes affect the system's states. It shows the system's operational behavior and user input reactions across its stages.



*Figure 5: State diagram showing various states for system.*

## **Activity Diagram**

A system's functioning is dynamically depicted in an activity diagram inside UML diagrams. It is a more advanced flowchart that shows how activities flow. The activity diagram in Figure 6 shows how system activities and processes happen.

From the user starting the game, the diagram follows events. As the user engages, a primary page appears with options. The user must decide whether to play game levels or change settings in this interactive framework.

The diagram shows a divergence from the main route if the user changes parameters. This creates a continuous interaction loop by returning to the main page.

If the user chooses a game stage, the diagram shows the advancement. The "play" button launches the playing screen. The diagram shows the transition from the primary screen to the interactive interface.

A significant event occurs after victory. The system plays music and displays high-quality images to improve the user experience. The diagram shows the game's resetting procedure if the user fails, allowing them to try again.

This comprehensive activity diagram shows the user's system journey by capturing the sequence of activities and decisions. It covers the interactive dynamics, choices, results, and reactions that shape the user experience.

A diagram of a game

Description automatically generated

*Figure 6: Activity diagram for system.*

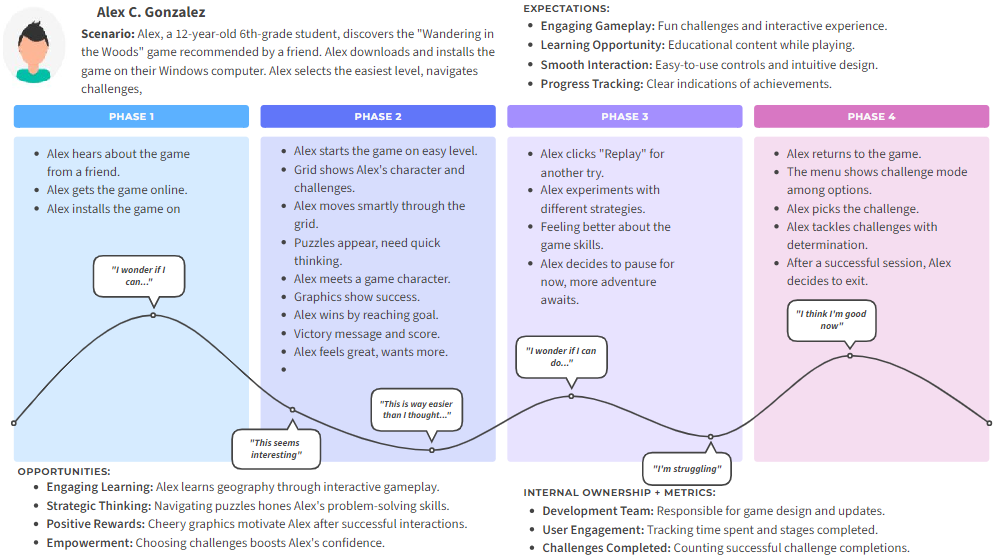
# **Customer Journey Map**

Customer journey maps, often called user journey maps, help brands understand customer interactions. This strategic exercise helps firms examine their operations from customers' perspectives, providing a comprehensive view of client interactions. Organizations may discover consumer problem areas and improvement possibilities using this technique.

A customer journey map visually depicts the whole customer experience with an organization, service, or product, from initial contact to final resolution. This detailed visualization captures the complex bond that develops on this journey.

An organization discovers and understands consumer demands, procedures, and perceptions using a customer journey map. This experiment reveals customers' emotions and thoughts at every interaction. This improved understanding helps refine strategy and improve client experience.

Design teams benefit greatly from customer journey maps. These maps help designers understand how customer expectations match brand offerings. They learn when expectations are fulfilled and exceeded and when improvements are needed. This insight drives iterative design and helps teams create customer-centric experiences.



*Figure 7: Visualizing the User Journey: 'Wandering in the Woods' Game Experience"*

# **Personas**

User personas are character profiles used by designers and developers to connect with a product's various consumers. Creators may put themselves in their consumers' shoes by creating personas. User personas help designers understand, identify with, and meet the requirements and expectations of different user groups.

User personas help designers and developers see users as unique persons with varying goals, interests, and obstacles. Personas are relatable archetypes that capture user types of features, actions, and motivations.

In a system study, designers carefully established a user persona, as seen in Figure 7. This persona represents an 14th-grader with academic weaknesses. This user also enjoys gaming. Thus, creating a game meets this persona's demands and may increase academic performance.

But this user persona may face new problems while playing the game. The game's interface should be simple to avoid complexity-related issues and offer a pleasant experience. Tips are crucial since they help overcome difficulties and provide a pleasant learning environment.



*Figure 8: Persons of a student for the system.*