**Course: Software Engineering**

**Project Name: Wandering in Woods**

**Summer – II**

**Design Document**

**August 26, 2022**

**Project Group 2**

**Table of Contents**

[1. Introduction 3](#_Toc112266724)

[1.1 Purpose 3](#_Toc112266725)

[1.2 Wandering in the Woods 3](#_Toc112266726)

[2. Process Model 4](#_Toc112266727)

[3. Use Case 4](#_Toc112266728)

[3.1 Use Case 1: Start Game 5](#_Toc112266729)

[3.2 Use Case 2: Play Game 5](#_Toc112266730)

[3.3 Use Case 3: Choose Stage 6](#_Toc112266731)

[3.4 Use Case 4: View Statistics 6](#_Toc112266732)

[3.5 Use Case 5: Replay Game 7](#_Toc112266733)

[3.6 Use Case 6: Exit Game 7](#_Toc112266734)

[3.7 Use Case 7: Move Diagonal 8](#_Toc112266735)

[3.8 Use Case 8: Move Random 8](#_Toc112266736)

[3.9 Use Case 9: Change Grid 9](#_Toc112266737)

[3.10 Use Case 10: Play Challenges 9](#_Toc112266738)

[3.11 Use Case 11: Test Variety of Wandering Methods 10](#_Toc112266739)

[4. UML Model 10](#_Toc112266740)

[4.1 Use Case Diagram 10](#_Toc112266741)

[4.2 Deployment Diagram 11](#_Toc112266742)

[4.3 Class Diagram 12](#_Toc112266743)

[4.4 State Diagram 16](#_Toc112266744)

[4.5 Activity Diagram 16](#_Toc112266745)

[5. Customer Journey Map 17](#_Toc112266746)

[6. Personas 18](#_Toc112266747)

# **1. Introduction**

## **1.1 Purpose**

This document will serve as the blueprint for the development of the Lost in the Woods game. Case study requirements will be examined, and results will be documented. For the sake of this guide, we will be using the agile methodology. Documentation for agile design is not static and is instead subject to change. In the first iteration, the model of the System Architecture must be defined. The design models should be defined when new data becomes available. The essential subsystems and components should be there, even if they are not complete. In line with Agile standards, this document contains just the information that the design team determined absolutely needed to be saved and updated. In our opinion, this paper represents a working draught of the final blueprint. Both the paper and the framework may be improved upon by using constructive feedback. The document helps the students to find this material useful in exploring the ideas underlying computer simulations. This prototype was designed to introduce elementary school children to the fundamentals of computing, mathematics, and computer programming.

A comprehensive explanation of the Wandering in the Woods game is presented herein. It will describe the system's goals and characteristics, as well as the many ways in which users may interact with the system. This document is aimed at the system's stakeholders as well as its designers.

## **1.2 Wandering in the Woods**

The game, "Wandering in the Woods," is a relatively simplistic visual aid meant to illustrate the core notion to elementary school children (and maybe entertain them). Students in grades three through five will be exposed to "little data," or information generated by one's own actions, in a manner that requires them to make decisions, evaluate data, and tackle further challenges. The sixth through eighth grade level will get a more advanced version that will require students to work with larger and smaller datasets, generate charts, and make decisions that significantly alter the data.

For the best results, the simulation should be run with two students sharing a single computer screen. The simulation must provide auditory suggestions and instructions, as well as tests to measure how well the students are completing the tasks.

# **2. Process Model**

Process models allow organizations to visualize how their internal business processes work to better manage and optimize them. In most cases, this is a continuous improvement exercise performed in an agile manner. The game will be designed using evolutionary process model which is the software development life cycle evolution model combines the iterative and incremental approaches. With the help of the evolutionary development model (EVO), products that better meet the demands of their target audience may be created at lower cost and with less risk, and incremental enhancements can be made to current products more quickly. The evolutionary approach recommends modularizing the project into manageable pieces, ranking them in order of importance to the client, and delivering them in stages. A huge number of chunks is being delivered to the customer, as well. During the entire project, the customer receives quantifiable goods and services, so he gains confidence. Changes in requirements are allowed through the model, and all work chunks are maintainable.

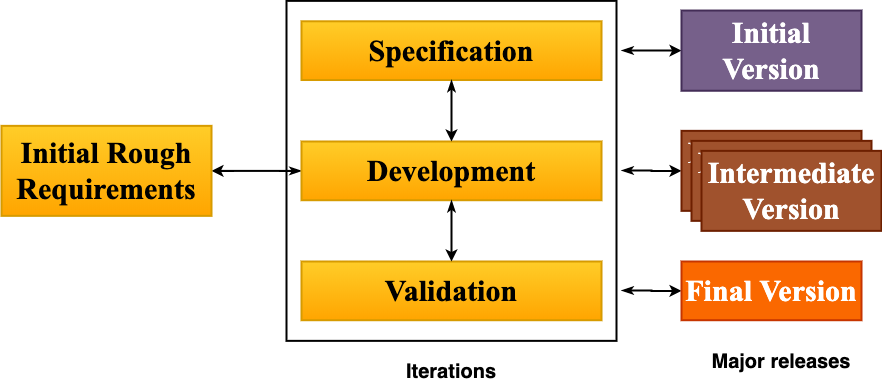


Figure 1: Evolutionary process model for system.

# **3. Use Case**

In use cases, users are described as performing specific tasks on system.  In it, the behaviour of a system is described based on the point of view of a user. In each use case, users' goals are the starting point, and the goal is achieved at the end. As well as explaining how the system should behave, use cases also help brainstorm what might go wrong.  To calculate the cost and complexity of a system, they provide a list of goals. Once the project team has determined which functions are required, they can negotiate which functions will be developed. The design scope of a use case should be considered when identifying elements within and outside of the process boundaries. A supporting actor or another use case should indicate anything outside the boundaries of the use case that is essential to its purpose. Systems, subsystems, or the entire organization can be the focus of a design. Businesses typically use the use cases to describe their business processes.

After the case study analysis, the design team created the following use cases and details are described below:

## **3.1 Use Case 1: Start Game**

|  |  |
| --- | --- |
| **Use Case ID** | 1 |
| **Description** | In this use case, the student will launch the game and will be navigated to main page of game and user will select start game. |
| **Actor** | Student |
| **Pre-Condition** | The student must install the game and must launch the game for starting the game. |
| **Flow of events** | 1. The student will click the game on screen, 2. The student will select the start game option. 3. The game will start. |
| **Post- Condition** | The system must start the game after loading the game in memory. |

## **3.2 Use Case 2: Play Game**

|  |  |
| --- | --- |
| **Use Case ID** | 2 |
| **Description** | In this use case, the student will play the game and will be navigated to the grid page where user will make moves. |
| **Actor** | Student |
| **Pre-Condition** | The student must select the play option from menu. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will press play option. 4. The system will display the game screen where grid is displayed. 5. The student will start playing. |
| **Post- Condition** | The system must display the grid page to student. |

## **3.3 Use Case 3: Choose Stage**

|  |  |
| --- | --- |
| **Use Case ID** | 3 |
| **Description** | In this use case, the student will select the stage as there are different stages for different students. |
| **Actor** | Student |
| **Pre-Condition** | The student must choose the stage. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will select choose stage option. 4. The student will press play option. 5. The system will display the game screen where grid is displayed. 6. The student will start playing. |
| **Post- Condition** | The system must display the grid page to student according to selected stage. |
| **Included Use Case** | Choose Stage |

## **3.4 Use Case 4: View Statistics**

|  |  |
| --- | --- |
| **Use Case ID** | 4 |
| **Description** | In this use case, the student will view the statistics according to the game being played. |
| **Actor** | Student |
| **Pre-Condition** | The student must play the game. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will select choose stage option. 4. The student will press play option. 5. The system will display the game screen where grid is displayed. 6. The student will start playing. 7. The system will display the statistics. |
| **Post- Condition** | The system must display the statistics from time to time. |

## **3.5 Use Case 5: Replay Game**

|  |  |
| --- | --- |
| **Use Case ID** | 5 |
| **Description** | In this use case, the student will replay the game multiple times. |
| **Actor** | Student |
| **Pre-Condition** | The student must play the game. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will select choose stage option. 4. The student will press play option. 5. The system will display the game screen where grid is displayed. 6. The student will start playing. 7. The system will display the statistics. 8. The student will replay the game. |
| **Post- Condition** | The system must replay the game upon request from student. |

## **3.6 Use Case 6: Exit Game**

|  |  |
| --- | --- |
| **Use Case ID** | 6 |
| **Description** | In this use case, the student will exit and end the game. |
| **Actor** | Student |
| **Pre-Condition** | The student must be inside the game screen. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game.    1. The student will exit game. 3. The student will play game.    1. The student will exit game. |
| **Post- Condition** | The system must exit the game and close the window of game. |

## **3.7 Use Case 7: Move Diagonal**

|  |  |
| --- | --- |
| **Use Case ID** | 7 |
| **Description** | In this use case, the student will make move in diagonally opposite direction. |
| **Actor** | K-2 Group Student |
| **Pre-Condition** | The student must be from k-2 group student. The student must start the game. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will press play option. 4. The system will display the game screen where grid is displayed. 5. The student will make move in diagonally opposite direction. 6. In case two students meet each other, the system will display happy graphic. 7. The system will reset the game. |
| **Post- Condition** | The system must count the number of moves and must store statistics. The system must also display high graphics and play music. |

## **3.8 Use Case 8: Move Random**

|  |  |
| --- | --- |
| **Use Case ID** | 8 |
| **Description** | In this use case, the student will make move in random direction and can place game characters anywhere on the grid. |
| **Actor** | K3-5 Group Student, K 6-8 Group Student |
| **Pre-Condition** | The student must be from k 3-5 or 6-8 group student. The student must start the game. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will press play option. 4. The system will display the game screen where grid is displayed. 5. The student will place game characters anywhere on grid. 6. The student will make move in any direction. 7. In case two students meet each other, the system will display happy graphic. 8. The system will reset the game. |
| **Post- Condition** | The system must count the number of moves and must store statistics. |

## **3.9 Use Case 9: Change Grid**

|  |  |
| --- | --- |
| **Use Case ID** | 9 |
| **Description** | In this use case, the student will change size of grid. |
| **Actor** | K3-5 Group Student, K 6-8 Group Student |
| **Pre-Condition** | The student must be from k 3-4 or 6-8 group student. The student must start the game. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will press play option. 4. The system will display the game screen where grid is displayed. 5. The student will change size of grid. 6. The student will make move in any direction. 7. In case two students meet each other, the system will display happy graphic. 8. The system will reset the game. |
| **Post- Condition** | The system must count the number of moves and must store statistics. The system must change the grid according to students’ input. |

## **3.10 Use Case 10: Play Challenges**

|  |  |
| --- | --- |
| **Use Case ID** | 10 |
| **Description** | In this use case, the student will play challenges presented from system. |
| **Actor** | K 6-8 Group Student |
| **Pre-Condition** | The student must be from k 6-8 group student. The student must start the game. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will press play option. 4. The system will display the game screen where grid is displayed. 5. The system will present new challenges to student on screen. 6. The student will make move in any direction. 7. In case two students meet each other, the system will display happy graphic. 8. The system will reset the game. |
| **Post- Condition** | The system must count the number of moves and must store statistics. |

## **3.11 Use Case 11: Test Variety of Wandering Methods**

|  |  |
| --- | --- |
| **Use Case ID** | 11 |
| **Description** | In this use case, the student will try and test different wandering methods for short time. |
| **Actor** | K 6-8 Group Student |
| **Pre-Condition** | The student must be from k 6-8 group student. The student must start the game. |
| **Flow of events** | 1. The student will click the game on screen. 2. The student will start the game. 3. The student will press play option. 4. The system will display the game screen where grid is displayed. 5. The system will test different wandering methods. 6. The student will make move in any direction. 7. In case two students meet each other, the system will display happy graphic. 8. The system will reset the game. |
| **Post- Condition** | The system must count the number of moves and must store statistics. The system must present wandering paths to student. |

# **UML Model**

## **Use Case Diagram**

Using use case diagrams, a system's dynamic aspect can be captured. UML use case diagrams are often used for analyzing various types of systems. This facilitates an understanding of how roles interact with each other in a system. System requirements can be gathered from use case diagrams by including both internal and external influences. Design requirements are most of these requirements. A system's functionalities are assembled via the preparation of use cases and the identification of actors. The use-case diagram describes a system's high-level functions and scope. These diagram helps in identification of actors and the interaction with one another. A use-case diagram describes what a system does and how it is used by actors but not how it operates internally. The use case of system is shown in Figure 2.

Diagram

Description automatically generated

Figure 2: Use case diagram for system.

## **Deployment Diagram**

A deployment diagram visualizes the hardware components of a system, their communications, and the software files that are placed on that hardware. An UML deployment diagram illustrates a system's physical architecture. A deployment diagram shows the relationship between hardware components and software components in a system as well as how the computing is physically distributed. The deployment diagram for system is shown in Figure 3. As system will be developed using Java so the java connectivity is used. The Apache server will be used as web server that will be used to take responses from client. The application server will use servlet application and will connect to SQL for database in which data will be stored.

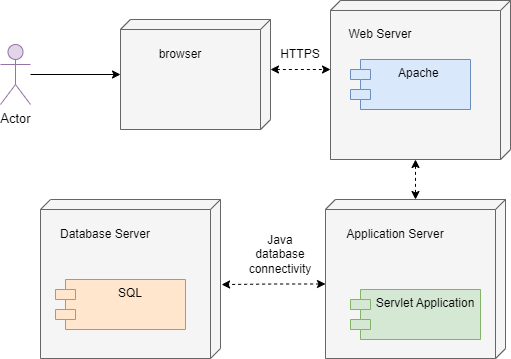


Figure 3: Deployment diagram for system.

## **Class Diagram**

Static diagrams are class diagrams. An application's static view is represented by it. A class diagram can serve as both a visual representation of, document for, and explanation of a system, as well as a tool for writing software code. The class for the system is shown in Figure 4. As there are three stages of game, so three classes are created and in each the functions are defined.

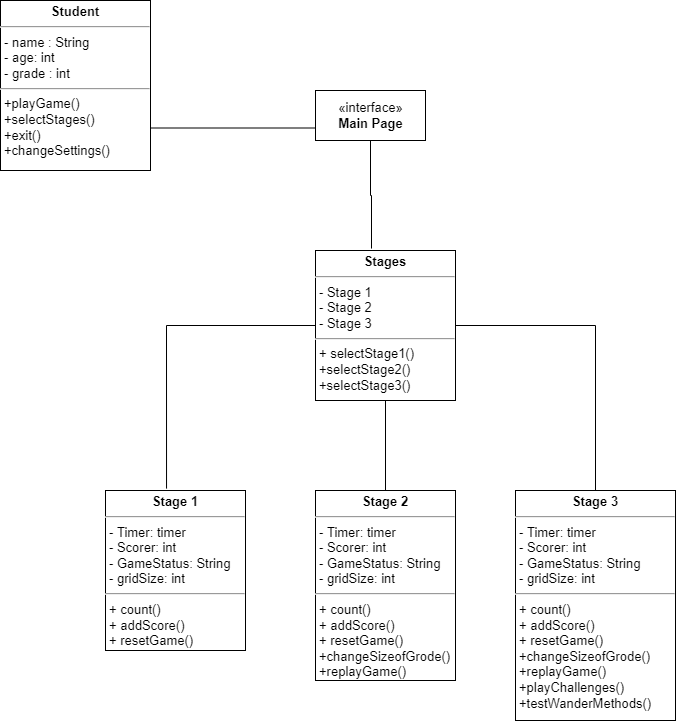


Figure 4: Class diagram for system.

**Description of classes:**

|  |  |
| --- | --- |
| **Student** | |
| **Attributes** | |
| **name** | Name of student in string |
| **age** | Age of student in integer |
| **grade** | Grade of student in integer to store the grade in which student is enrolled. |
| **Operations** | |
| **Playgame()** | Function to play game |
| **Selectstages()** | Function to select stages in game |
| **Exit()** | Function to exit the game |
| **Changesettings()** | Function to change settings of game |

|  |  |
| --- | --- |
| **Stages** | |
| **Attributes** | |
| **Stage 1** | The information about stage 1 is stored in it. |
| **Stage 2** | The information about stage 2 is stored in it. |
| **Stage 3** | The information about stage 3 is stored in it. |
| **Operations** | |
| **SelectStage1()** | Function to select stage 1 of game. |
| **Selectstage2()** | Function to select stage 2 of game. |
| **Selectstage3()** | Function to select stage 3 of game. |

|  |  |
| --- | --- |
| **Stage 1** | |
| **Attributes** | |
| **timer** | The time to store each move of student |
| **scorer** | To store score of each student. |
| **Gamestatus** | To store status of game as there are different states like play, resume, exit. |
| **Gridsize** |  |
| **Operations** | |
| **Count()** | Function to count the moves of student. |
| **Addscore()** | Function to add the scores of students on each move. |
| **Resetgame()** | Function to reset the game. |

|  |  |
| --- | --- |
| **Stage 2** | |
| **Attributes** | |
| **timer** | The time to store each move of student |
| **scorer** | To store score of each student. |
| **Gamestatus** | To store status of game as there are different states like play, resume, exit. |
| **Gridsize** |  |
| **Operations** | |
| **Count()** | Function to count the moves of student. |
| **Addscore()** | Function to add the scores of students on each move. |
| **Resetgame()** | Function to reset the game. |
| **Changesizeof grid()** | Function to change the size of grid |
| **Replaygame()f** | Function to replay the game |

|  |  |
| --- | --- |
| **Stage 3** | |
| **Attributes** | |
| **timer** | The time to store each move of student |
| **scorer** | To store score of each student. |
| **Gamestatus** | To store status of game as there are different states like play, resume, exit. |
| **Gridsize** |  |
| **Operations** | |
| **Count()** | Function to count the moves of student. |
| **Addscore()** | Function to add the scores of students on each move. |
| **Resetgame()** | Function to reset the game. |
| **Changesizeof grid()** | Function to change the size of grid |
| **Replaygame()f** | Function to replay the game |
| **Playchallenges()** | Function to present the new challenges to grade 6-8 group students |
| **Testwandermethods()** | Function to try and test different methods of wandering |

## **State Diagram**

System state diagrams show the state of a system or part of a system over time. The behavior is represented using finite state transitions in this behavioral diagram. A state diagram is also known as a state machine or state chart diagram. It is common to use these terms interchangeably. State diagrams serve to describe how a class responds to changing external stimuli and time over time. All classes have states, but we don't use State diagrams to model all classes. The states should be modeled with at least three states. The state diagram of system is shown in Figure 5. When the user starts the game, the game will be in “READY” state. Then after pressing playing option the state will change to “PLAY” state. If user pause the game, the state will change to “PAUSE” state. On resuming the state will change to “RESUME” state. On exiting game, the state will change to “END” state.

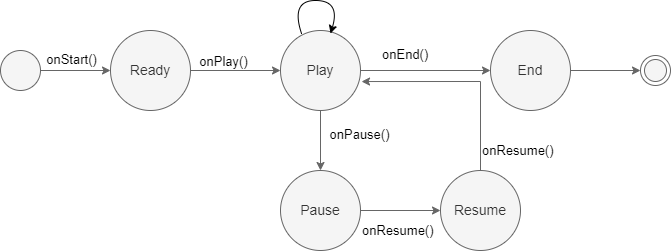


Figure 5: State diagram for system.

## **Activity Diagram**

An activity diagram describes dynamic aspects of a system in UML diagrams. An activity diagram is basically an advanced form of flowchart that represents how activities flow from one to the other. The activity diagram of system is shown in Figure 6. The user will start the game. A main page with options will be displayed. The user will have choice whether to select game stages or change settings. If user will select change settings option, he can again go back to main page. If user selects the stage, then user will press play button and will move to playing screen. If the user wins the system will play music and high graphics will be displayed. Else the game will reset.

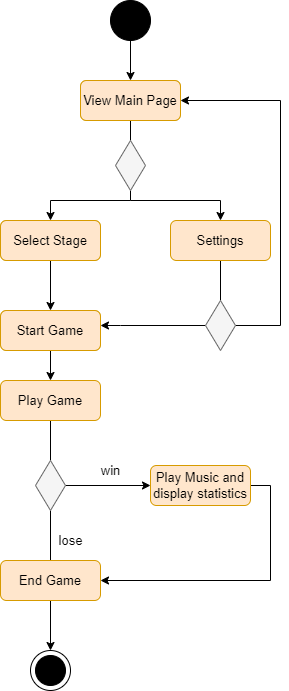


Figure 6: Activity diagram for system.

# **Customer Journey Map**

Customers' journey maps (also called user journey maps) demonstrate how their interactions with your brand work. Businesses can use this exercise to view their business through the eyes of their customers. Using it allows you to find out what customers are most unhappy with and how to improve them. The customer journey map illustrates the relationship between a brand, service, and product from start to finish. An organization can gain a better understanding of its customer's needs, processes, and perceptions by creating a journey map. Design teams use customer journey maps to learn how customer expectations are met and determine what needs to be improved.

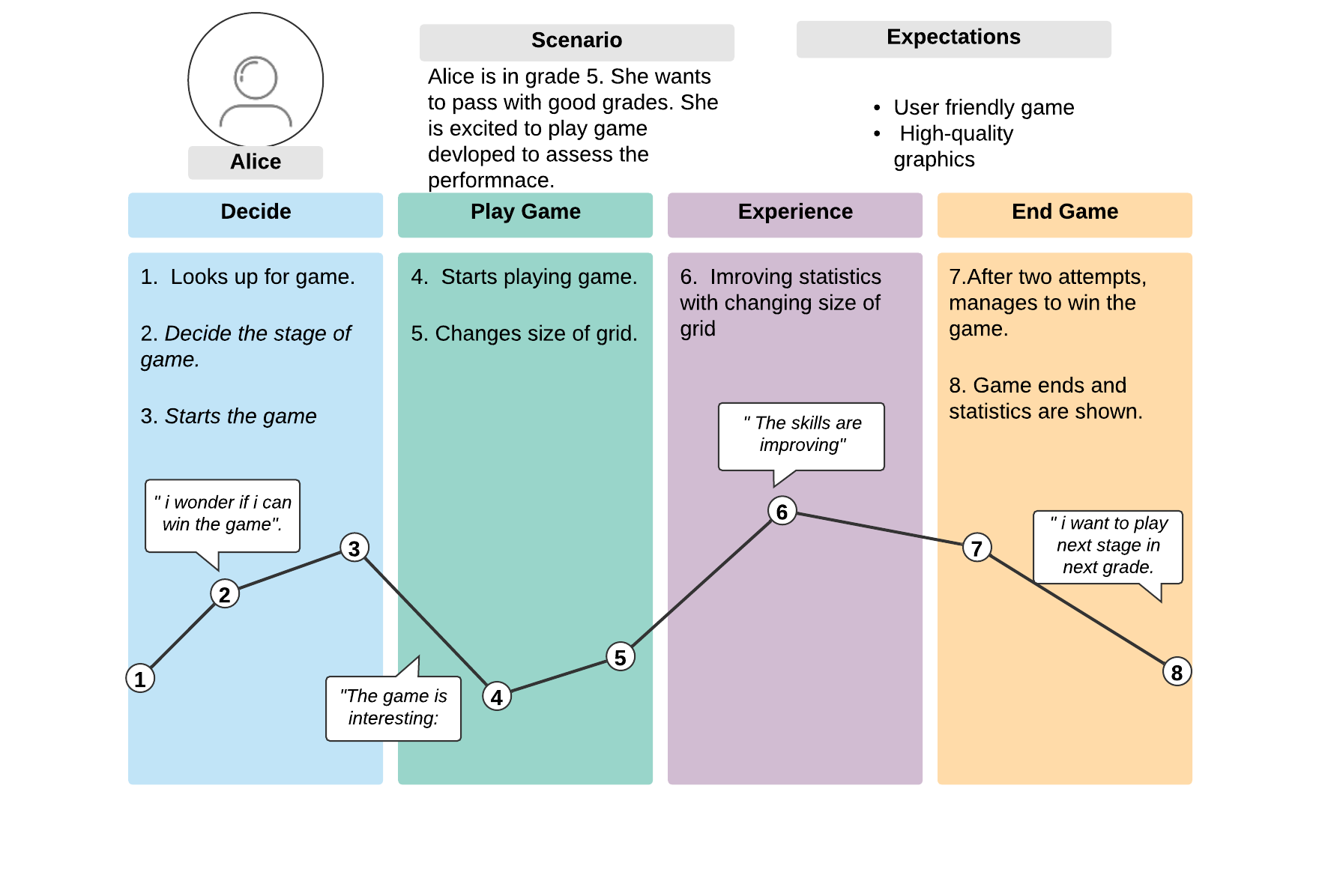


Figure 7: customer journey map of user of system.

# **Personas**

User personas are archetypes of users that designers and developers can use to empathize with them. Creating personas can help you step out of yourself. In addition to helping, you understand and identify with the needs and expectations of different users, designing for them can help you recognize their differences. After analyzing the system, the designers have identified a user as shown in Figure 7, who is student of 8 grade and is weak in some subjects. He is fond of playing games. So, this game will help him a lot. But he may face some new challenges in game, so hints should also be provided, and interface should be easy to navigate.

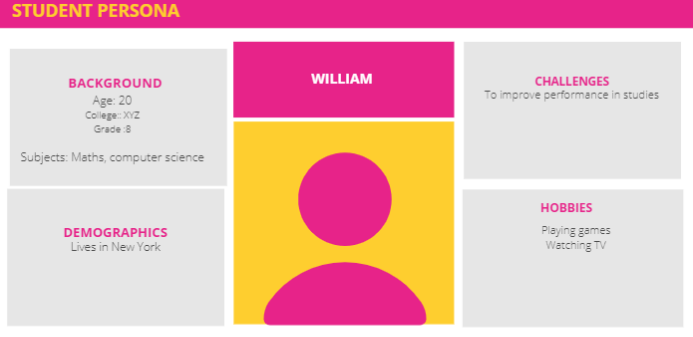


Figure 8: Persons of a student.