Design Documentation

CS2212 Group 2 - Design Documentation

Version	Date	Author (s)	Summary of Changes
User Interface Mock 1.0 Development Environment 1.0 File Formats 1.0 UML Class Diagram 1.0	Wednesday February 14th, 2024 - Tuesday February 20th, 2024	• Hailey • Maneet • Simran • Victor	• User Interface Mockup ○ Discussed what type of screens our application will have ○ How many wireframes there should be • Development environment ○ Discussed what kind of development environment our team will be using • File format ○ Discussed how and what file format we will use to store the data • UML Class Diagram ○ Discussed what classes should be implemented
Introduction 1.0 User Interface Mock Up 2.0 Pattern 2.0	Wednesday February 21st 2024 - Tuesday February 27th 2024	• Simran • Hailey • Victor • Kareena • Maneet	• Introduction ○ Objectives (Hailey) • Worked on wireframes (User interface Mockup) ○ Login Screen (Simran) ○ Creating Account (Simran) ○ High Scores List (Simran) ○ Main Menu (Hailey) ○ Game Play (Hailey) ○ Tutorial (Hailey) ○ Achievements (Victor) ○ Choosing difficulty of new game (Victor) ○ Progress/ results (Victor) ○ Debug/ level selection (Victor) ○ Instructor Dashboard (Maneet) ○ Setting Dropdown Menu (Kareena) • Patterns ○ Described and explained what patterns are should be implemented in the project and how we will implement them (Simran) ■ Provided descriptions for: • Builder, Prototype, Object Pool, Visitor, Chain of Responsibility, Adapter patterns • UML Class Diagram - Kareena
File Formats 2.0 Patterns 2.0 UML Class Diagram 2.0 Development Environment 2.0 Summary 1.0	Wednesday February 28th, 2024 - Monday March 4th, 2024	• Simran • Maneet • Hailey • Victor • Kareena	• Added descriptions of the wireframes in UI Mockup section - Maneet • Added changes to File Format Description - Maneet • UML Class Diagram - Kareena ○ Created general classes (Player, Instructor, and Data collect user's score information, and time taken for each gameplay) ○ Brainstormed some methods, getters, and setters needed for each class ○ Made a general framework that will be shared with the group to add on ○ Added methods and attributes for each class (Instructor, Player, and Data) ○ Added User interface and game play, leaderboards, and explained in description purpose of PlayerObserver and LevelBuilder interface and GamePlay class (Hailey) ○ Added PlayerObserver and LevelBuilder interfaces (Simran) • Patterns ○ Described and explained what patterns are should be implemented in the project and how we will implement them ■ Added on and provided descriptions for: • Added descriptions for: Builder, private class data, and observer patterns (Simran) • Added some changes (Victor) • Overview (Maneet) • Development Environment ○ Described what language, commenting documentation, and testing approach will be used (Maneet) ○ Described what game development library, integrated development environment (IDE), and graphical user interface (GUI) will be used - Simran • Summary (Hailey) • References (Simran)

Table 1.0 Tabular Revision History

2. Introduction

2.1 Overview

Capital Word Chain is an educational game to serve the purpose of educating players on geography, more specifically, the capital cities and states in North America. The primary focus is to educate the player in a more appealing way by combining the educational aspect with entertainment. The software's design is in the best interest of the user to provide them with the best possible experience. The user interface is structured in a proper manner, where the user is able to login or create an account first, and then being provided with many options to go about. The user is also given a reward system to keep the user engaged. The use of design patterns will aid us in being able to provide the user with a well thought out system. System requirements have been well thought-out and are motivated by the desire to provide quality. An outlined version of the system requirements can be seen in Section 3 (Class Diagrams). The UML diagram will help us in creating a high quality software for the user. Capital Word Chain, an educational experience for users, will be designed in a thoughtful approach considering the users' needs.

2.2 Objectives

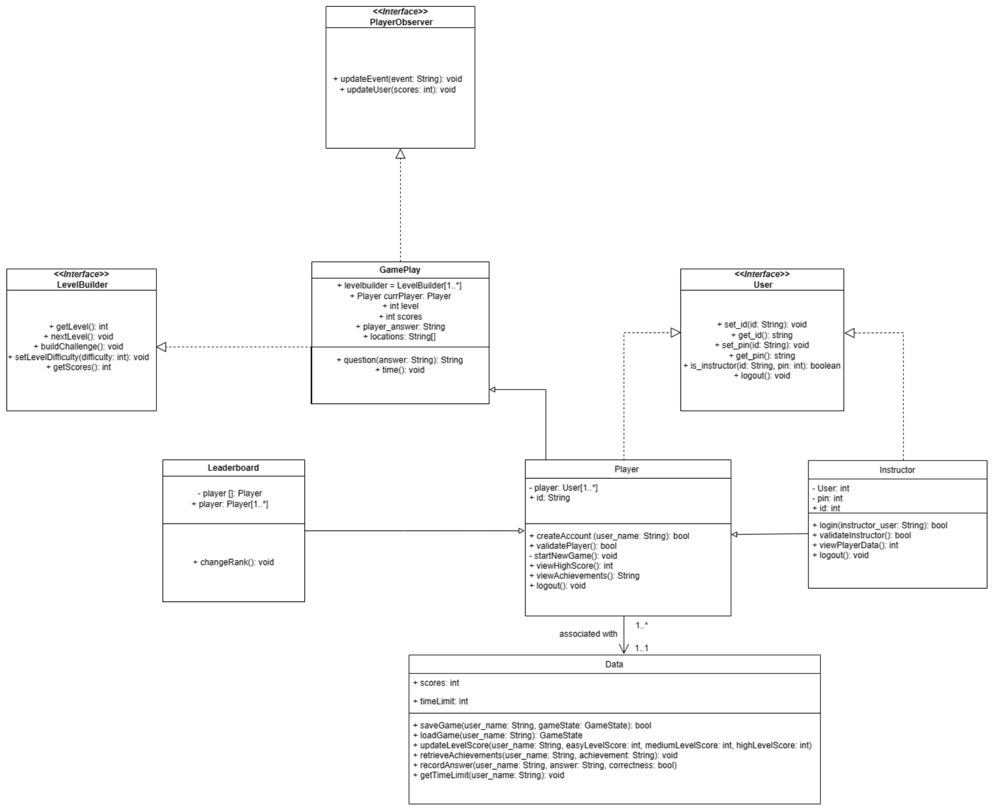
This document serves as a design outline to help us forward with drafting and implementing the project. We included UML for a visual representation of the project's nascent design. The UML includes specification of attributes inclusive of their respective types and visibility parameters, and the detailing of methods, incorporating parameters, parameter types, return types, and visibility attributes. The UML diagram is anticipated to serve as a narrative, documenting associations, generalizations, multiplicity, and any requisite notations to distinctly clarify the interrelationships among classes. As we embark on the programming stage, this UML representation will serve as an invaluable guide for seamless project execution.

2.3 References

- Source Making. "Design Patterns and Refactoring." *Sourcemaking.com*, 2019, sourcemaking.com/design_patterns.
- "The Catalog of Design Patterns." *Refactoring.guru*, The Catalog of Design Patterns (refactoring.guru)
- Servos, D. (2024). CS2212 Group Project Specification - Version 1.0 - Winter Session 2024. OWL. CS2212 Group Project Specification.pdf (uwo.ca)

3. Class Diagrams

- **https://drive.google.com/file/d/1f_3jw_M1SC0Bzs4dNSR_4yLhWz5UAaYb/view?usp=sharing**



<<Interface>>
PlayerObserver
+ updateEvent(event: String): void
+ updateUser(scores: int): void
<<Interface>>
LevelBuilder
+ getLevel(): int
+ nextLevel(): void
+ buildChallenge(): void
+ setLevelDifficulty(difficulty: int): void
+ getScores(): int
GamePlay
+ levelbuilder = LevelBuilder[1..*]
+ Player currPlayer: Player
+ int level
+ int scores
+ player_answer: String
+ locations: String[]
+ question(answer: String): String
+ time(): void
<<Interface>>
User
+ set_id(id: String): void
+ get_id(): string
+ set_pin(id: String): void
+ get_pin(): string
+ is_instructor(id: String, pin: int): boolean
+ logout(): void
Leaderboard
- player []: Player
+ player: Player[1..*]
+ changeRank(): void
Player
- player: User[1..*]
+ id: String
+ createAccount (user_name: String): bool
+ validatePlayer(): bool
- startNewGame(): void
+ viewHighScore(): int
+ viewAchievements(): String
+ logout(): void
Instructor
- User: int
- pin: int
+ id: int
+ login(instructor_user: String): bool
+ validateInstructor(): bool
+ viewPlayerData(): int
+ logout(): void
associated with
1..*
1..1
Data
+ scores: int
+ timeLimit: int
+ saveGame(user_name: String, gameState: GameState): bool
+ loadGame(user_name: String): GameState
+ updateLevelScore(user_name: String, easyLevelScore: int, mediumLevelScore: int, highLevelScore: int)
+ retrieveAchievements(user_name: String, achievement: String): void
+ recordAnswer(user_name: String, answer: String, correctness: bool)
+ getTimeLimit(user_name: String): void

(Note: The preview of the UML diagram in the draw.io doesnt show all of the updates, however when you click on open with draw.io, you can see the updated version).

Figure 12.0 UML Class Diagram

Player and Instructor classes implements **User interface** which consists of getter and setter methods for basic information(ID, PIN) for instructor and player.

The **player** class of the diagram includes two attributes: user and ID. The main objective of this class is to ensure that the player class maintains the data of a player user. It keeps the player's game login username to enter the player's game dashboard and begin the game; moreover, it stores the player's ID to validate the player's user. In the player class, the following methods exist: createAccount(), validatePlayer(), startNewGame(), viewHighScore(), viewAchievements(), and logout(). These methods do the following:

- createAccount(): Allows the player to create a new account in the word chain game
- validatePlayer(): Checks the validity of a player's credentials for logging into the game
- startNewGame(): Enables the player to initiate a new game session
- viewHighScore(): Displays the high scores achieved by each player for each level
- viewAchievements(): Showcases the achievements unlocked by each player
- logout(): Enables the player to exit their game session, ending their current session.

The **instructor** extends Player class and uses interface from User class. Instructor class of the diagram includes three attributes; user, pin, and id. The main goal of the instructor class is to be able to hold the data of an instructor user. It stores the instructor users' username to login, pin code to enter the instructor's dashboard, as well as an id to identify the user. The class also includes methods login, validateInstructor, viewPlayerData, and logout. All of these methods provide the instructor with the ability to enter into the system and be able to retrieve the student/player data, as well as exit out of the system with the progress saved.

The **data** class of the diagram includes two attributes: scores, timeLimit. The main goal of the data class is to store the user's gameplay data. It stores the user's game information through the methods: saveGame(), loadGame(), updateLevelScore(), retrieveAchievements(), recordAnswer(), getTimeLimit. These methods do the following:

- saveGame(): Saves the current state of the game progress
- loadGame(): Loads a saved game
- updateLevelScore(): Updates the score for the current level of the game
- retrieveAchievements(): Retrieves the achievements unlocked by each player
- recordAnswer(): Records the player's answers to a question in the wordchain game
- getTimeLimit(): Retrieves the time limit set for each complete task in each level of the game

The **leaderboard** class includes attributes player [] and player. The main goal of the leaderboard class is to hold the high scores of the top players. The attribute player [] represents an array of all players, and player (public) represents each Player object created. The class also includes the method changeRank which will adjust and update the leaderboards once a new high score is achieved which allows the player to move up on the leaderboards or be displayed if they are not already on the leaderboards.

The **LevelBuilder** interface includes 5 methods: getLevel, nextLevel, buildChallenge, setLevelDifficulty, and getScores. The main goal of this interface is to create a LevelBuilder object that will allow GamePlay class to retrieve level information of the user's chosen level. It consists of following methods:

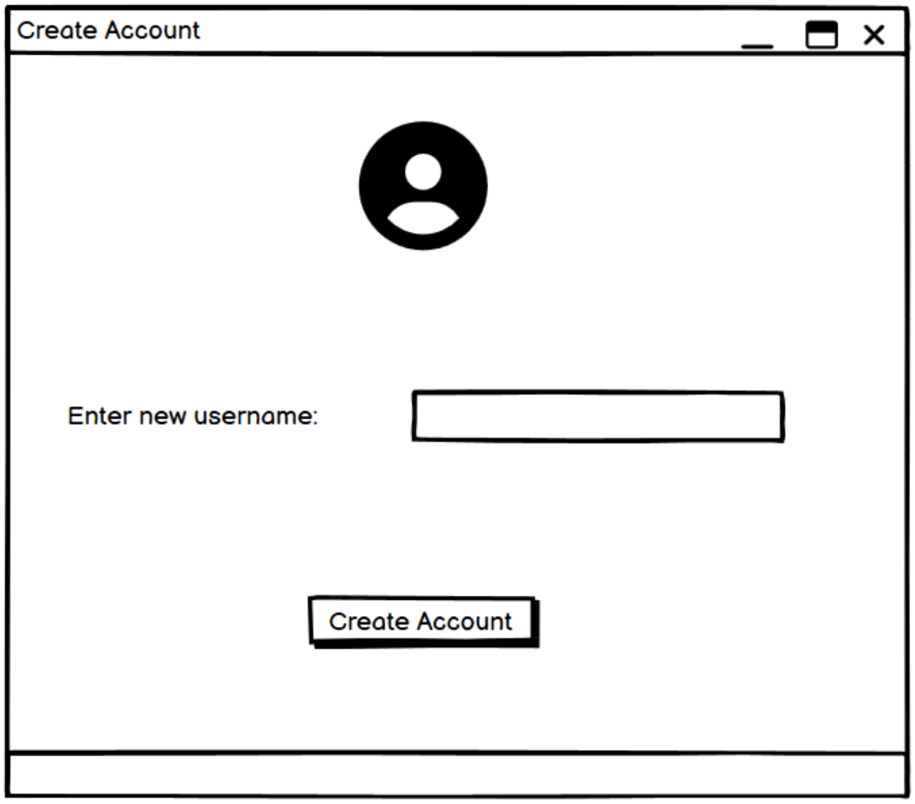
- getLevel() - Get the current level chosen by the user.
- nextLevel() - Move to the next level in the sequence.
- buildChallenges() - Build challenges for the current level.
- setLevelDifficulty() - Set the difficulty level for the current level.
- getScores() - Retrieve scores related to the current level.

PlayerObserver is an interface for that will allow class GamePlay to update information such as scores and and events during the world chain game play.

GamePlay class will implement methods that will mainly focus on the play aspects of world chain. It uses objects/methods from the Player class.

- LevelBuilder levelBuild - An instance of the LevelBuilder interface responsible for managing the progression and content of word chains.
- Player curPlayer - An instance of the Player class representing the current player participating in the word chain game.
- int level - An integer indicating the current level of the word chain game.
- int scores - An integer representing the accumulated scores achieved by the player during the game.
- player_answer - An integer representing the accumulated scores achieved by the player during the game.
- locations: Strings[] - An array of strings storing data about the word chain vocabularies. In this context, it represents American States and cities.
- question (String answer) - A method that returns the correct answer for the current word chain. If the player's input matches this answer, their score will be increased.
- time() - A method of the LevelBuilder interface that sets the time limit for each word chain. The player is expected to provide their input within this time limit.

4. User Interface Mockup



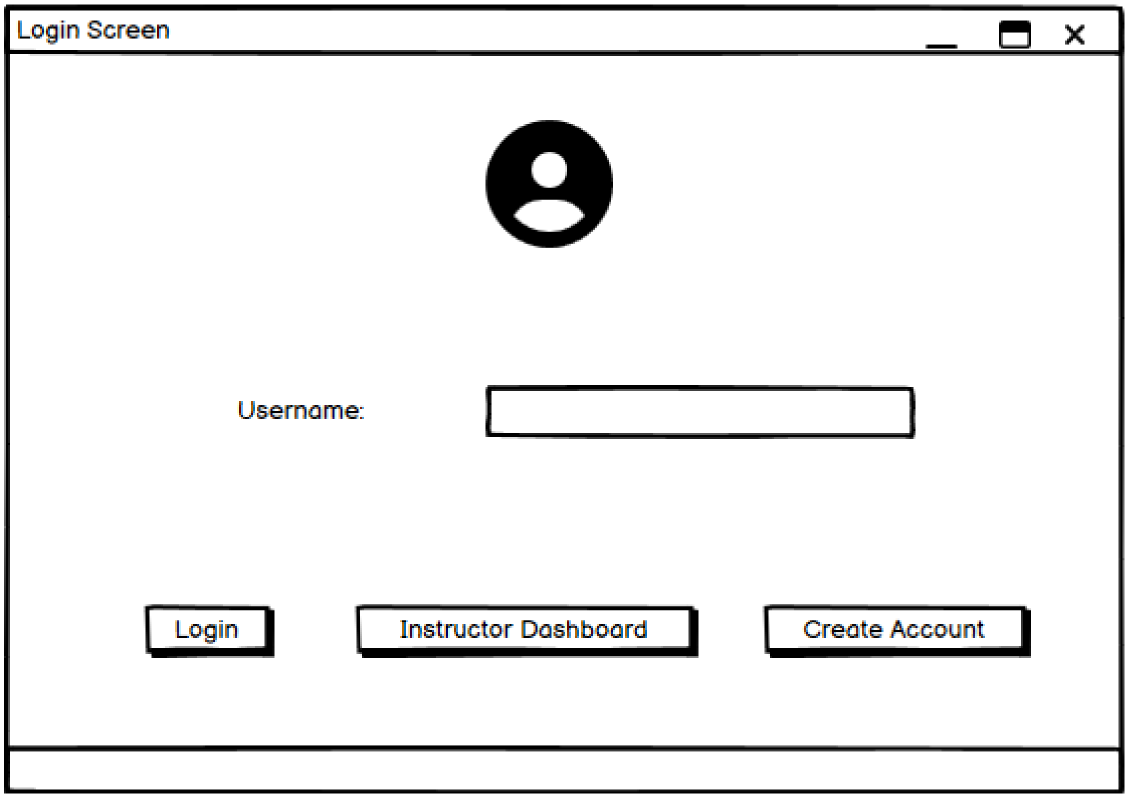


Figure 1.0 Login and Account Creation Screen

Users can log in by entering their username in the designated text field. If they don't have an account, they can initiate the account creation process by clicking the "Create Account" button. This action will redirect them to the create account page, as illustrated in the wireframe below. On the create account page, users can establish a new account by entering an unused username and selecting the "Create Account" button. Upon successful account creation, they will be automatically redirected to the main menu. This initial page serves as the entry point before transferring users to the main menu (Figure 2.0).

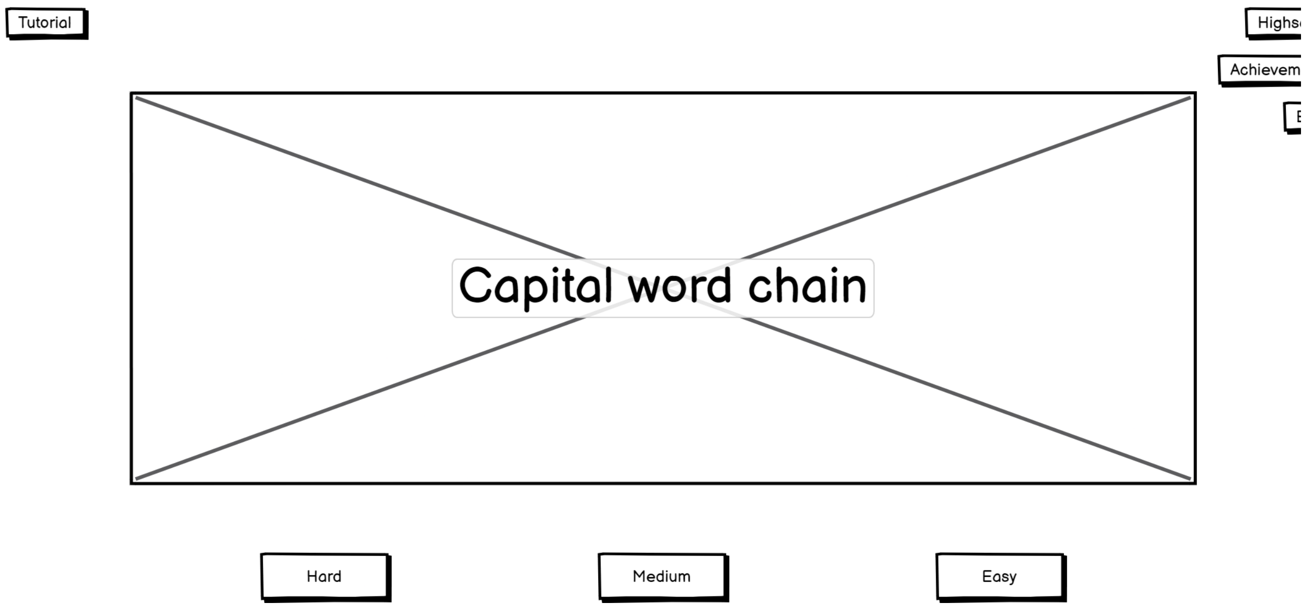


Figure 2.0 Main Menu (Starting Screen)

The main menu is designed with essential buttons such as "New Game," "Load Previous Games," "Quit" (exit), "User’s Scores," and a "Tutorial" for first-time players. Additionally, we've incorporated an achievements section as part of our extended functionality. In this section, users can view the rewards and achievements they have acquired throughout their gameplay experience (Figure 3.0). The inclusion of these features enhances the overall user interface and engagement within the application.

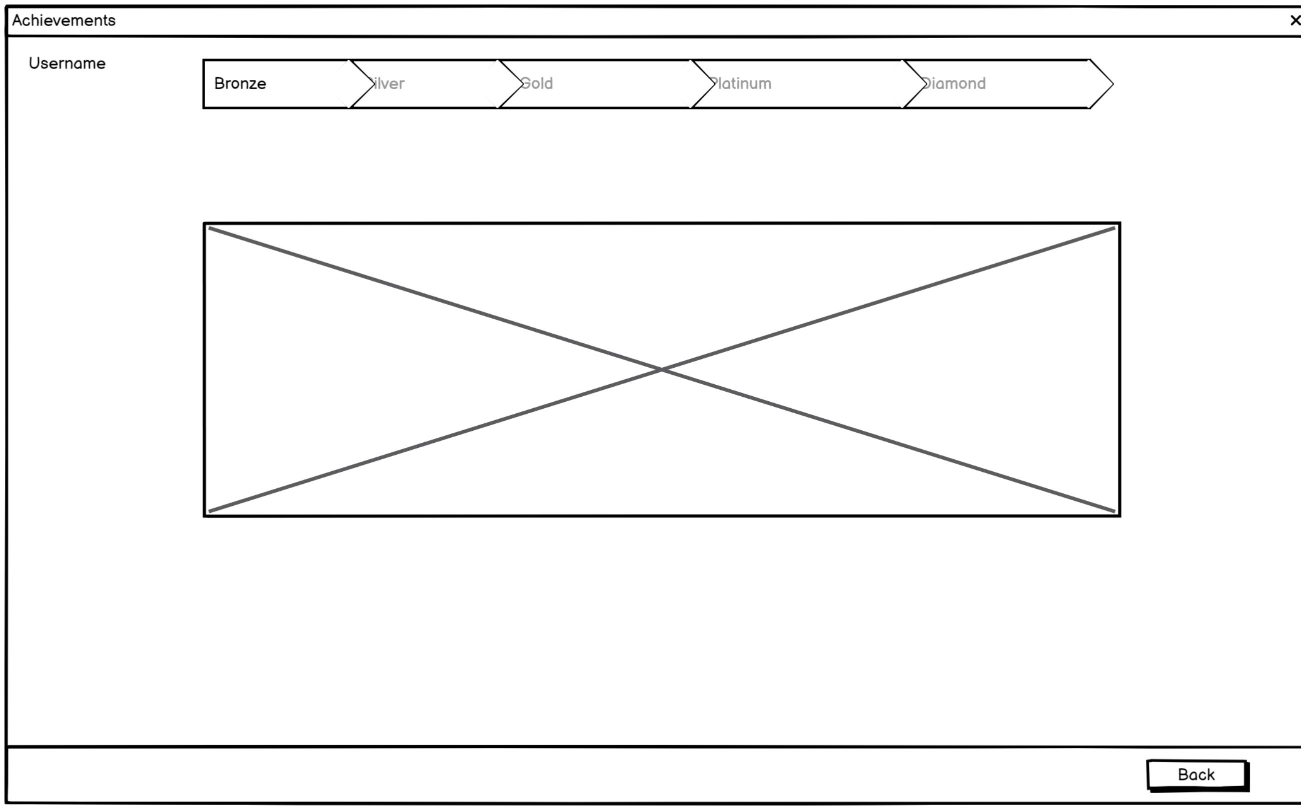


Figure 3.0 Achievements

The Achievements page serves as a progress tracker, showcasing the user's advancements in the game. It functions akin to ranks, with achievements categorized into bronze, silver, and subsequent levels. This tiered structure provides users with a tangible sense of accomplishment and serves as a rewarding system. As users attain different levels of achievements, they can visually gauge their progress and enjoy a satisfying gaming experience enriched by the recognition of their in-game accomplishments.

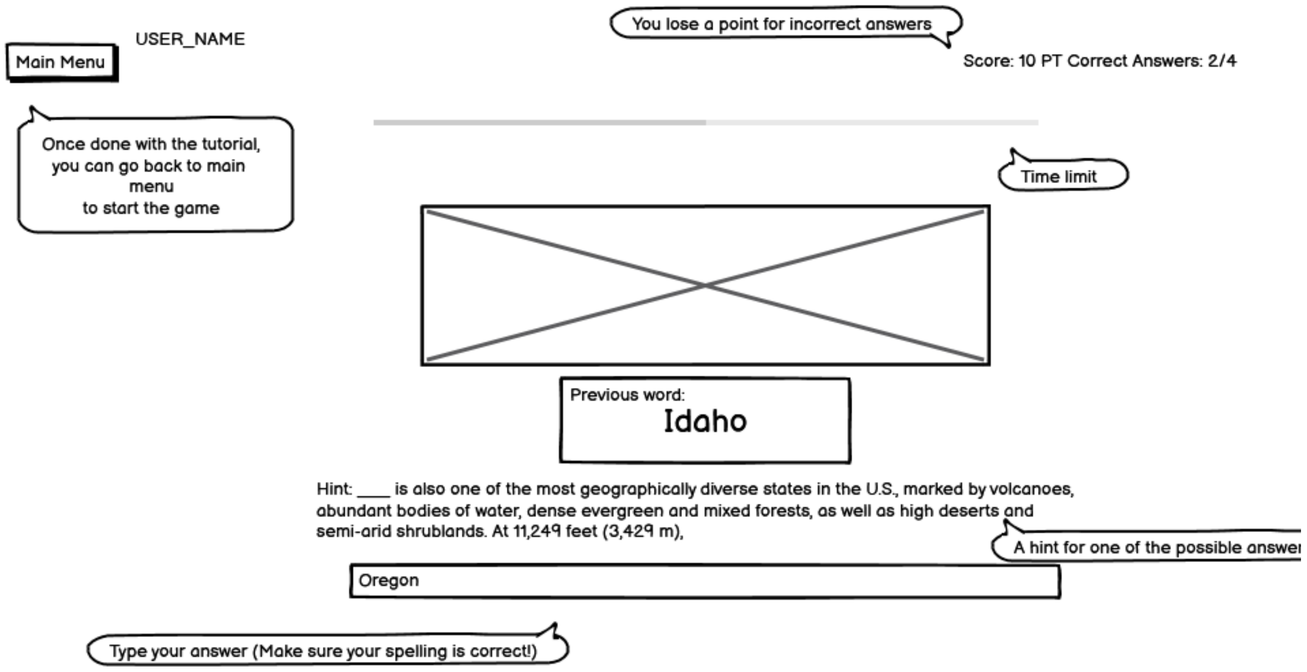

USER_NAME
Main Menu
You lose a point for incorrect answers
Score: 10 PT Correct Answers: 2/4
Once done with the tutorial, you can go back to main menu to start the game
Time limit
Previous word:
Idaho
Hint: ____ is also one of the most geographically diverse states in the U.S., marked by volcanoes, abundant bodies of water, dense evergreen and mixed forests, as well as high deserts and semi-arid shrublands. At 11,249 feet (3,429 m),
A hint for one of the possible answe
Oregon
Type your answer (Make sure your spelling is correct!)

Figure 4.0 Instructional/tutorial

The Instructions/Tutorial page offers a comprehensive guide, presenting step-by-step insights into the game. The tutorial explains the layout of the gameplay page and the game's objective, and details the scoring method. Each instruction is revealed individually, prompting users to either left-click or press the space bar to progress to the next explanation. To enhance understanding, practical examples of gameplay are displayed on the screen, providing users with a visual reference and ensuring a thorough comprehension of the game mechanics before embarking on their gaming experience.

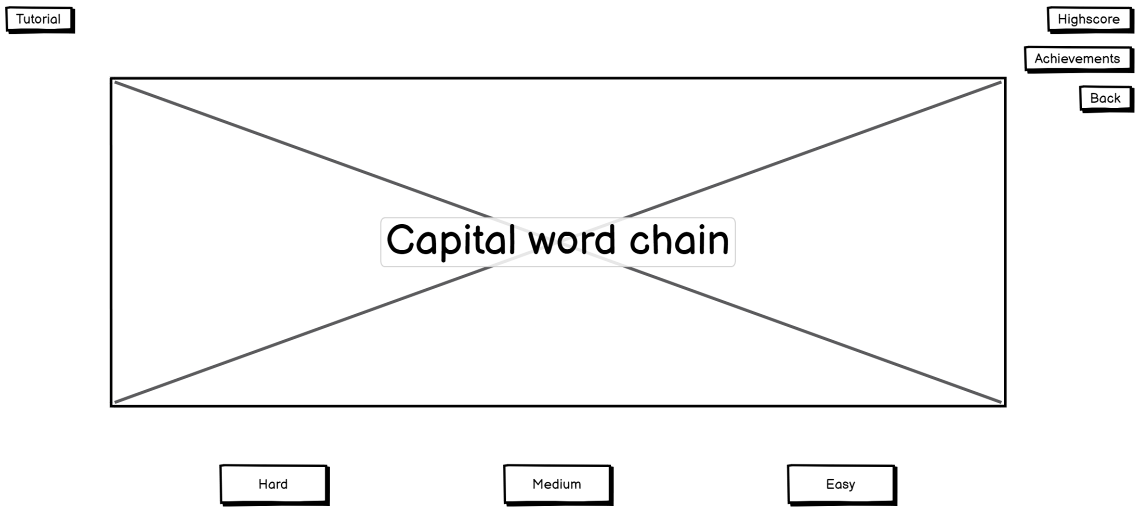


Figure 5.0 Choose difficulty of new game

This page facilitates the user in choosing the difficulty mode for the game, offering options such as hard, medium, and easy modes. Once the user makes their selection, they are transitioned into the game environment. For added flexibility, a back button is provided, allowing users to return to the main menu at any point if they wish to make changes or explore other features. This design ensures a user-friendly experience, allowing players to tailor their gaming preferences before diving into the gameplay.

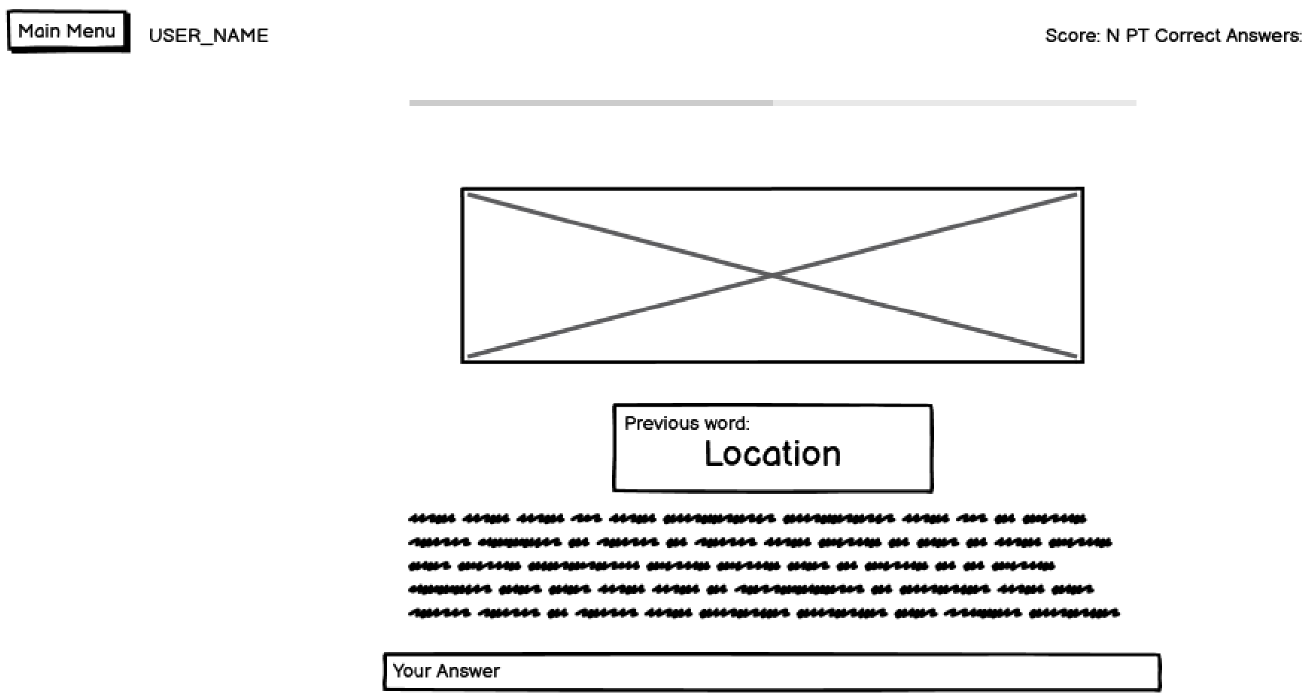


Figure 6.0 Game Play

The Game Play Screen is the central interface where users actively engage in gameplay. It features an image of a map, accompanied by the display of the previous word to guide users in constructing their answers. Below this, a helpful hint is presented to assist users in formulating the correct response for the ongoing chain. A timer prominently positioned above the image indicates the remaining time for users to submit their answers.

The top right corner showcases the user's current score and the number of correct answers achieved thus far, providing a real-time overview of their performance. For added convenience, users can find an option in the top left corner to return to the main menu (Figure 2.0), though it is important to note that this action will result in the loss of current gameplay. This setup ensures an immersive and informative gameplay experience, with quick access to key information and controls.

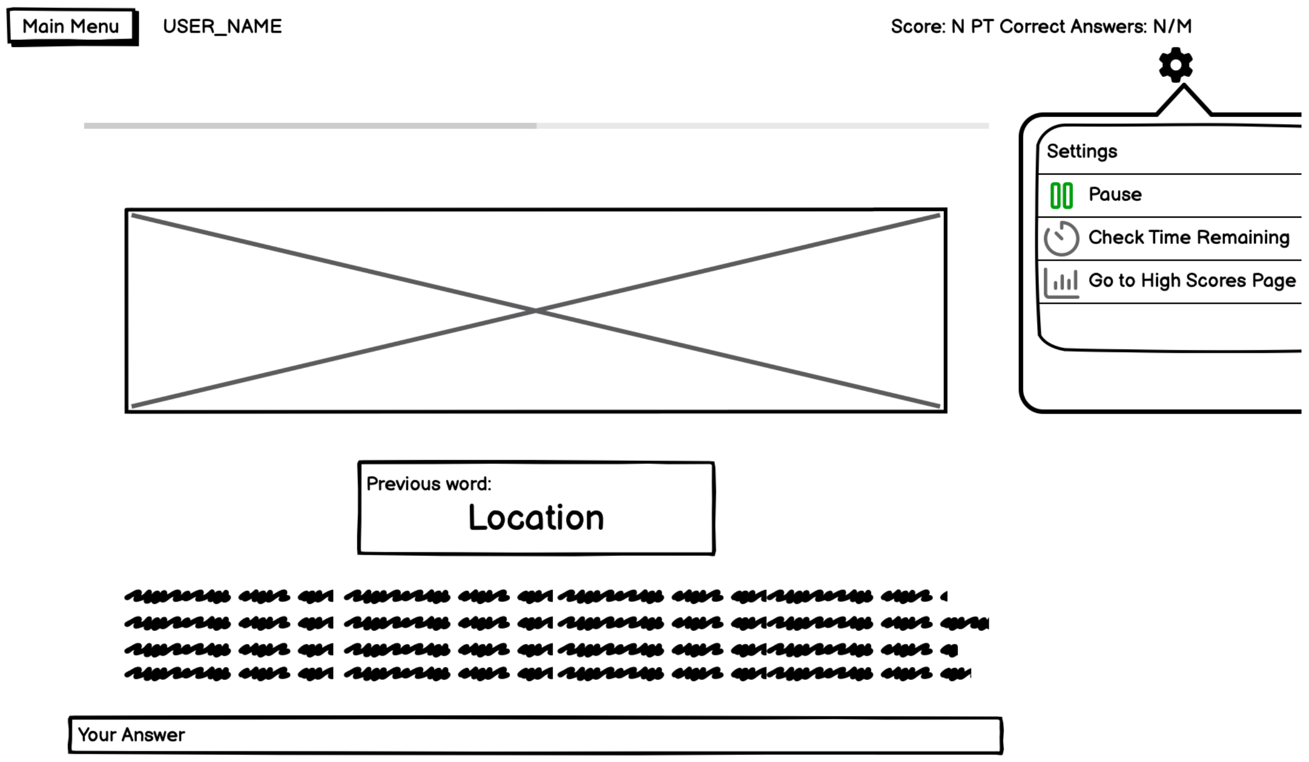
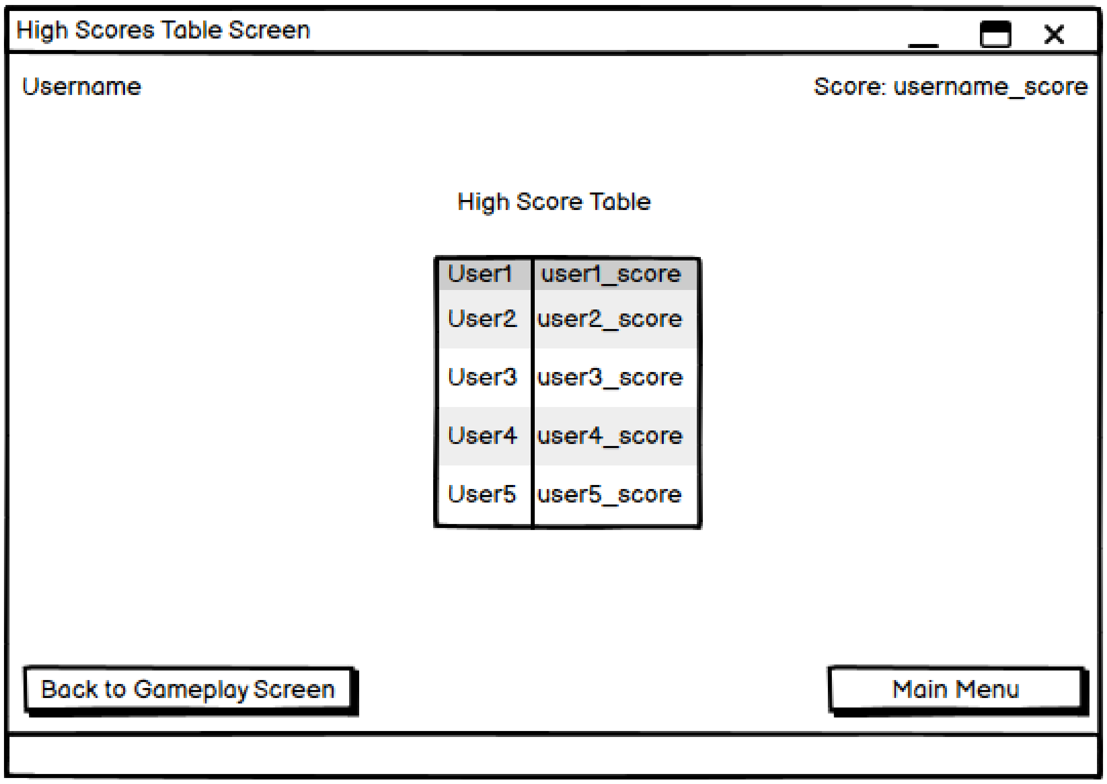


Figure 7.0 Settings Dropdown Menu

The settings drop down button can allow the user to pause the game if they need to at any moment to leave and come back mid-game, without losing their progress. The user is able to check the time remaining in a numerical format rather than viewing the bar if they wish to see a more precise time. The user can also navigate to the leaderboards/high scores page through here.



User1	user1_score
User2	user2_score
User3	user3_score
User4	user4_score
User5	user5_score

Figure 8.0 High score list

The High Scores list, also known as the leaderboards, exhibits the names of top-performing players who have achieved the highest scores in the game. In the right corner, the user's own score is displayed, providing them with immediate feedback on their current standing in comparison to the top players. Users have the flexibility to return to their ongoing gameplay session if they accessed the page through the game. Alternatively, they can opt to navigate back to the main menu. This design ensures that users can easily track their progress, assess their performance against other players, and make informed decisions about their next actions within the game.

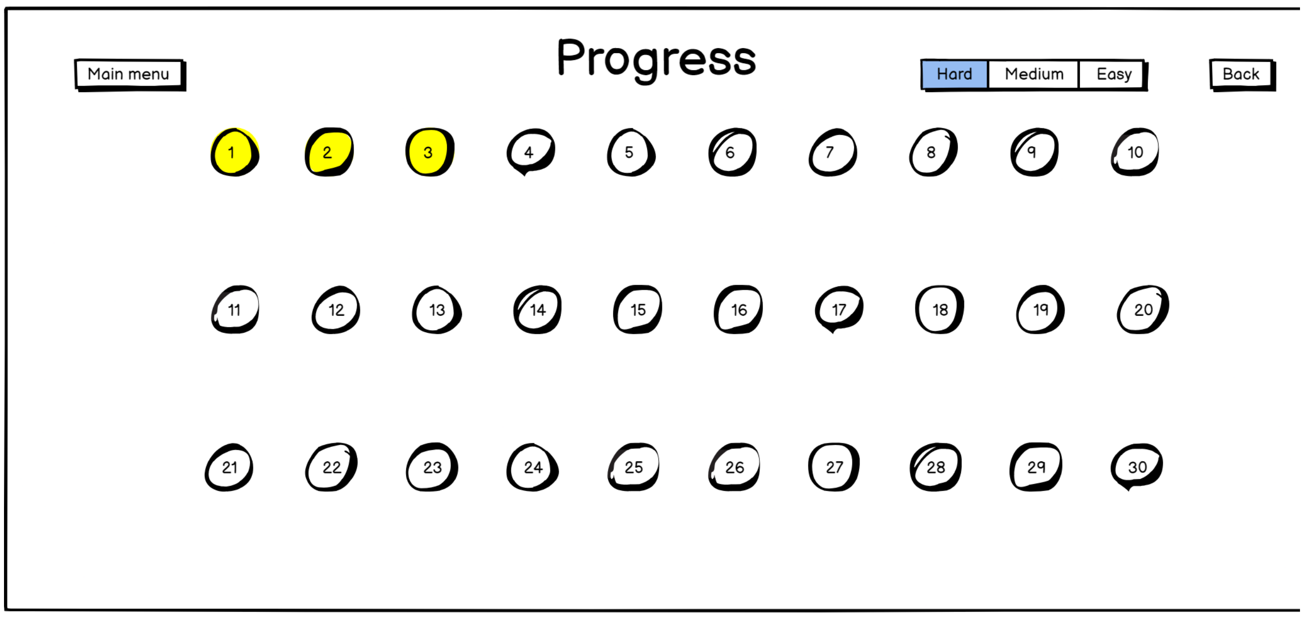


Figure 9.0 Progress/results (simply a screen that shows the results of the problem/level/stage/challenge)

The progress results page is showed at end of the game so the user can view their entire progress completed. The yellow/highlighted circles are the levels completed and the plain ones are the ones that are yet to be completed. The user is able to go to the main menu as well, and also go back to the game by pressing back.

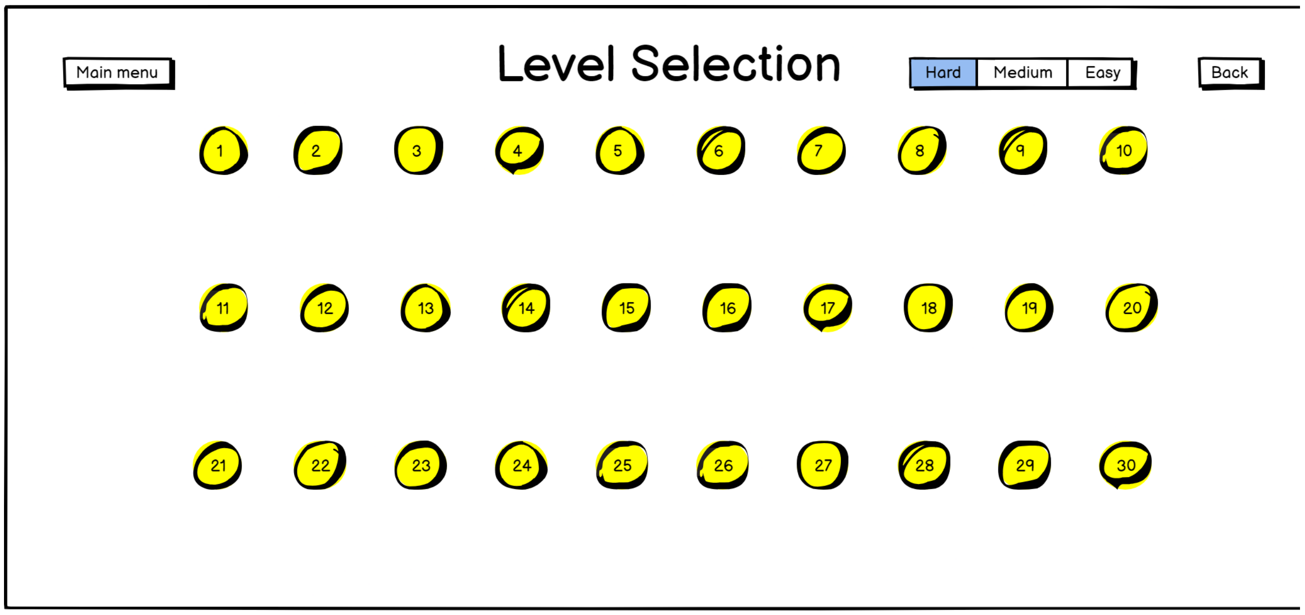
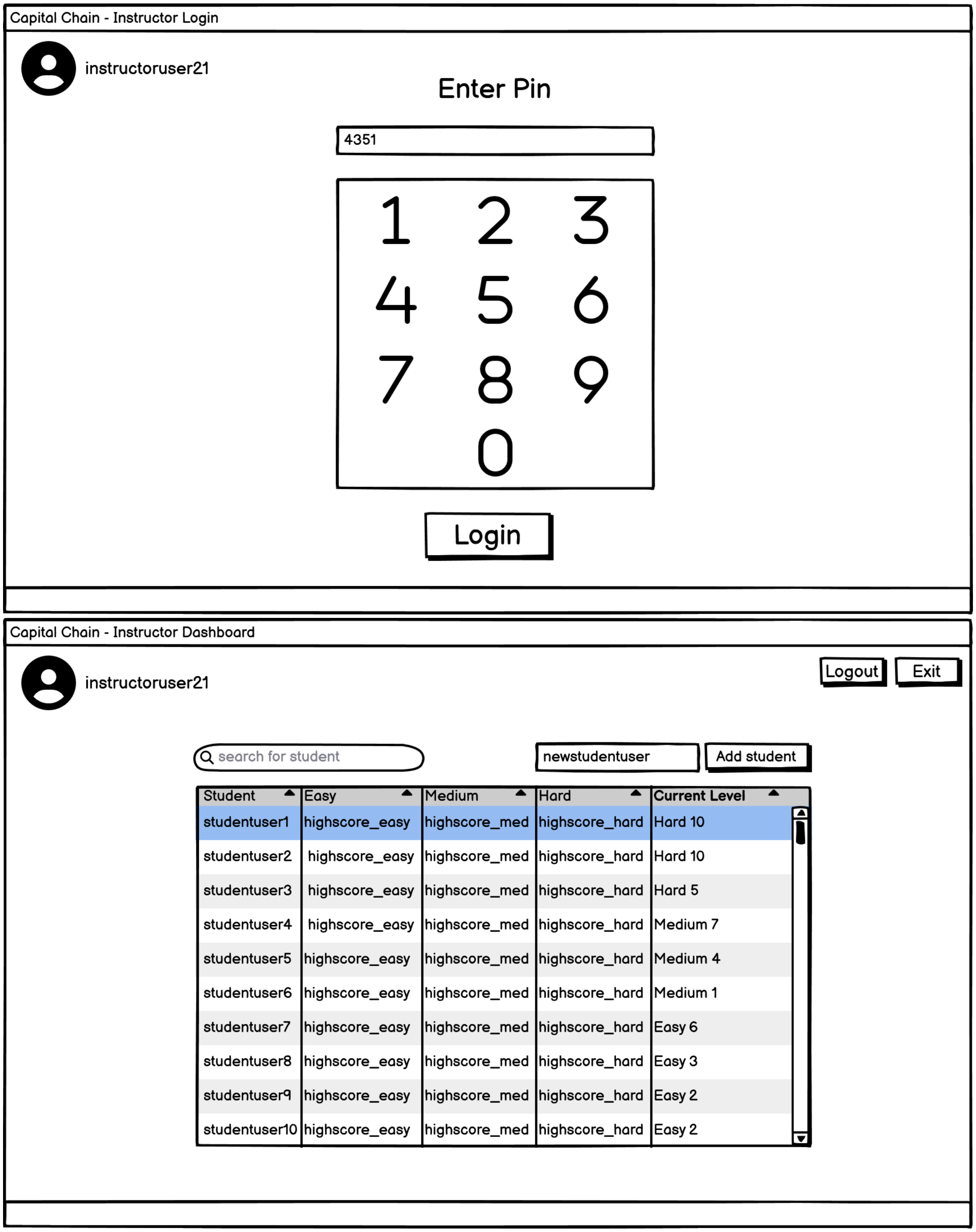


Figure 10.0 Debug/ level selection

The admin is able to view all levels for testing the game. They will be able to enter into any stage without previous completion/progress unlike a regular user. This page will be hidden to the admin only, and will be protected by a password only the admin will know.



Capital Chain - Instructor Login

instructoruser21

Enter Pin

4351

Capital Chain - Instructor Dashboard

instructoruser21

newstudentuser

Student	Easy	Medium	Hard	Current Level
studentuser1	highscore_easy	highscore_med	highscore_hard	Hard 10
studentuser2	highscore_easy	highscore_med	highscore_hard	Hard 10
studentuser3	highscore_easy	highscore_med	highscore_hard	Hard 5
studentuser4	highscore_easy	highscore_med	highscore_hard	Medium 7
studentuser5	highscore_easy	highscore_med	highscore_hard	Medium 4
studentuser6	highscore_easy	highscore_med	highscore_hard	Medium 1
studentuser7	highscore_easy	highscore_med	highscore_hard	Easy 6
studentuser8	highscore_easy	highscore_med	highscore_hard	Easy 3
studentuser9	highscore_easy	highscore_med	highscore_hard	Easy 2
studentuser10	highscore_easy	highscore_med	highscore_hard	Easy 2

Figure 11.0 Instructor Dashboard

Instructor Login Page: Access to student data and information is safeguarded through a personal identification number (PIN). Instructors must enter their PIN to securely view student details.

Instructor Dashboard: This page features a table displaying all students within the instructor's "classroom." The table offers a concise summary of each student's high scores across different difficulty levels, along with their current progress level. Clicking on a student provides access to a detailed progress/results page, mirroring the student view for a comprehensive understanding of individual performance. The dashboard allows instructors to search for students by username and facilitates student addition by entering the username and utilizing the "add student" button, integrating them into the overview table.

5. File Formats

We will be using CSV file format to store the data in our game. Mainly we will be using the file to store the student/user information as well as their scores. If it is a student user, the information stored will be in the format of: Username, Highest_Score, Easy_Score, Medium_Score, Hard_Score, Current_Stage, Instructor_User. The instructor user is optional as not all users will be apart of a group/class with an instructor. The data for a user without an instructor will be filled by a symbol that can be extracted. This data can also be used to display the information in the instructor dashboard. For an instructor user, the information will be stored in the format of: Instructor_User, Pin. This will keep the pin code to access instructor dashboard connected with the instructor. To handle the file format, we will be user the libraries Java Scanner and BufferedReader. With BufferedReader we can read the CSV files and then use Scanner to extract the data into the system. A preview of how the data will be stored (in a tabled format) can be seen below.

Username	**Highest_Score**	**Easy_Score**	**Medium_Score**	**High_Score**	**Current_Stage**	**Instructor_User**

Instructor_User	**Pin**

6. Development Environment

We will be using Java to create our game. Javadoc will be used to document our code, and JUnit will be used to test. We will also use Java swing as graphical user interface tool, libGDX as a game development library which we will implement in our project and Eclipse as our integrated development environment.

7. Patterns

- Builder
    - The builder pattern constructs complex objects step by step, which is useful for our application for creating different player profiles and game levels. This will be helpful for players strategically building their chain of capitals, Ottawa to Austin. The separation of the pattern of the construction process, it allows for extensive customization and flexibility in the game design. We plan to apply this pattern through an implementation of a LevelBuilder interface, which demonstrates the component-based construction part of game levels, allowing for clarity and expandability of the application . The implementation of the builder pattern and its impact on our application will be demonstrated in our class diagram (to show its impact on the system architecture), showing its role in enhancing maintainability and coherence. This approach, detailed in the following URL: Builder (refactoring.guru), aligns with our goal of delivering an engaging learning experience with its impact on class design reflected in our class diagrams to maintain clarity.
- Private Class Data
    - The private class data design pattern is used to minimize the exposure of class diagrams and protect other objects' integrity. It also encapsulates the data attributes within a single class and manages its access details (held in a private inner class). We plan to apply this pattern through an implementation of Instructor class, where the Instructor class includes such data which has a personal identification number (pin) being held in a private inner class. Only public methods of Instructor class can access this data ensuring only public authorized methods can read these attributes. This approach reduces risk of any modifications of the data which may be unauthorized. When implementing this pattern, it reduces the class's public interface which simplifies the usage of the class. This approach, detailed in the following URL: Private Class Data (sourcemaking.com), aligning with our goal of protecting the Instructor's pin so that the instructor can access instructor dashboard while other users cannot.

- Observer
    - The observer pattern uses a one to many dependency between objects so that when an object changes its state the other dependent objects are notified and automatically updated. This pattern is useful when an event source needs to notify many event listeners regarding a change in the state or the events that occurred. We plan to apply this pattern so that it would update the players about changes in the game state, like updates about the score, any change in the levels or achievements. We will implement this through the GamePlay class, where there will be instances of the PlayerObserver interface with a update method. Whenever the GamePlay state changes, it will loop though the registered observers and call its update method with the associated event data. this approach, detailed in the following URL: Observer Design Pattern (sourcemaking.com), aligning with our goal to update events in our application.

8. Summary

The design document outlines the development of "Capital Word Chain," an educational game centered on North American capital cities and states. It integrates education with entertainment, prioritizing user experience through a well-thought-out interface and a reward system. The document provides a comprehensive overview of the project's objectives, including visual aids such as UML diagrams to guide the implementation phase effectively. The User Interface Mockup section details various interfaces, including login screens, main menus, achievements, tutorials, and game play screens. Each interface is described and visually presented for a clear understanding. In summary, the design document provides a holistic view of the Capital Word Chain project, covering its objectives, references, class diagrams, user interface mockups, file formats, development environment, and the utilization of various design patterns to achieve a sophisticated and engaging educational game.

Terms	Definitions
libGDX	An open-source game development framework written in Java, with some parts in native code. It is designed for the development of 2D and 3D games, and it supports multiple platforms, including desktop, Android, iOS, and web-based applications. libGDX simplifies game development by providing a set of features and tools that handle common tasks, such as graphics rendering, input handling, and asset management.
Java swing	A set of graphical user interface (GUI) components for building desktop applications in Java. Developed to provide a more sophisticated and modern GUI toolkit than the earlier Abstract Window Toolkit (AWT), Swing is part of the Java Foundation Classes (JFC) and is included in the Java Standard Edition (SE).
JUnit	A widely used open-source testing framework for Java programming language. It provides a standardized way to write and execute test cases for Java applications, ensuring that code behaves as expected and facilitating automated testing practices. JUnit follows the xUnit architecture and has become a standard in the Java development community.
Javadoc	A documentation generator tool used for generating API documentation in HTML format from Java source code. It is an integral part of the Java Development Kit (JDK) and allows developers to embed documentation comments directly in their Java code. These comments are then processed by the Javadoc tool to create comprehensive and readable documentation for classes, interfaces, methods, and other code elements.
Java	A high-level, versatile, and object-oriented programming language.
Buffered Reader	A BufferedReader is a class in Java, commonly used to read data from a source in a buffered manner. It provides a more efficient way to read characters, arrays, or lines from a character-input stream, such as a file or an input stream. The buffering mechanism reduces the number of I/O operations, improving the overall performance of reading operations.
Java Scanner	In Java, the Scanner class is part of the java.util package and is used for parsing primitive types and strings. It provides methods to read input of different types from various sources, such as the console, files, or strings.

Table 2.0 Terminology