|  |  |
| --- | --- |
| VILNIAUS KOLEGIJA/UNIVERSITY OF APPLIED SCIENCES  FACULTY OF ELECTRONICS AND INFORMATICS  SOFTWARE ENGINEERING DEPARTMENT | |
| A picture containing icon  Description automatically generated | |
| **Software Engineering** | |
| Project  Software Engineering 6531BX028 PI21E | |
| STUDENTS | VITOLD SKUDER  NAZAR LAVKART  SEBASTIAN TERMEN |
| (SIGNATURE)  2023-12-11 | |
| LECTURER J. Katina | |
| (SIGNATURE) 2023-12-11 | |
|  | |
| 2023 | |

**CONTENTS**

[INTRODUCTION 3](#_Toc153244315)

[1. FORMULATING THE TASK 9](#_Toc153244331)

[**1.1. Analysis of similar systems** 9](#_Toc153244332)

[**1.2 Functional requirements** 10](#_Toc153244333)

[**1.3 Non-functional requirements** 11](#_Toc153244334)

[2. TASK ANALYSIS 13](#_Toc153244335)

[**2.1. Use case diagram and description** 13](#_Toc153244336)

[**2.2 Activity diagram(s) and description (s)** 19](#_Toc153244337)

[**2.3 The entity-relationship diagram and its description** 24](#_Toc153244338)

[**2.4 Layout of the user interface** 25](#_Toc153244339)

[3. SOFTWARE IMPLEMENTATION 27](#_Toc153244340)

[**3.1. System architecture and description** 28](#_Toc153244341)

[**3.2. System files and description** 29](#_Toc153244342)

[**3.3. Description of classes and their methods** 30](#_Toc153244343)

[4. INSTALLATION AND USER INSTRUCTIONS 33](#_Toc153244344)

[**4.1. Dependencies on other software products** 34](#_Toc153244345)

[**4.2 Computer hardware parameters** 34](#_Toc153244346)

[**4.3 Deployment Schedule** 34](#_Toc153244347)

[**4.4. Launch description** 34](#_Toc153244348)

[**4.5 Removal steps** 34](#_Toc153244349)

[**4.6 Instructions for use** 34](#_Toc153244350)

[CONCLUSIONS AND PROPOSALS 35](#_Toc153244351)

[LIST OF SOURCES OF INFORMATION 37](#_Toc153244352)

# **INTRODUCTION**

**Origin**

The origin of our project lies in the intersection of classic gaming and modern technology. We suggest a deep engagement with both the historical aspects of classic games like Tic Tac Toe, Checkers, and Connect Four, and the challenges of reimagining them for the digital era. This project was initiated to bridge the gap between nostalgic board games and the digital gaming landscape, appealing to both traditional game enthusiasts and new audiences.

**Aim and objectives**

The primary aim of this project is to create a digital platform in a form of a framework that not only revives classic games for a contemporary audience but also enhances them with modern technology and design principles. This objective is to be achieved through a meticulous process testing with a deep focus on quality, user experience, and technological robustness. Our objectives include:

1. **Research and Analysis:** Studying the evolution of these games and existing digital adaptations, as indicated by our extensive documentation.
2. **Requirements Formulation:** Define clear functional and non-functional requirements for each game.
3. **Theoretical and Practical Development:** Combining the theoretical models with practical development strategies, as seen in our Source and Library directories.
4. **Implementation with Advanced Tools:** Utilizing advanced development tools and technologies, as suggested by the use of Raylib and CMake, to build engaging and efficient game designs.
5. **Comprehensive Testing:** Following a detailed test plan to ensure optimal functionality and user experience.
6. **Documentation and User Assistance:** Creating user manuals and guides.

This approach reflects a balanced mix of traditional game appreciation and modern software development practices.

For the development of our classic games suite, we utilized a diverse range of tools and technologies.

* **Devices Used:** Our primary development was carried out on computers, although the specific hardware specifications are varied among the team members.
* **Software and Development Environments:**
  + We used sophisticated development tools like VSCode, JetBrains Rider IDE.
  + The Tools folder contains CPM and a PowerShell script (TestRunner.ps1), for package management and scripting for testing and deployment.
  + CMake was the primary buildsystem to deploy and test our C++ project.
* **Programming Languages and Technologies:**
  + Our primary and only programming language is C++ we also have a bit of SQL.
  + The durlib external library, developed by Sebastian Termen, to manage user input/output, parsing, and logging.
  + Raylib is the main rendering engine used to display graphics to the user.
  + GTest was used to develop tests.
  + SQLite was used to manage tiny amounts of user data.

This setup reflects a blend of traditional and modern development practices, suitable for creating a robust and scalable framework for classic grid-based games.

**Project promoters**

Our project is being developed by a team of three students, each bringing unique skills and responsibilities to the table. Below is a breakdown of each team member's role and responsibilities.

**Table 1. Breakdown of project responsibilities**

|  |  |
| --- | --- |
| **Student** | **Responsibilities** |
| Vitold Skuder | Project Manager, Analyst |
| Nazar Lavkart | Programmer, Architect, Tester |
| Sebastian Termen | Project Manager, Programmer, Designer, Tester |

**Table 2. Responsibilities of the project functions**

|  |  |
| --- | --- |
| **Responsibilities** | **Responsibilities** |
| Project Manager | * Project evaluation * Creating a project plan * Project supervision |
| Analyst | * Analyzing requirements * Researching game market trends * Ensuring the alignment of project goals with user needs |
| Programmer | * Writing and debugging code * Integrating different game components * Developing algorithms for game logic |
| Architect | * Designing the system architecture * Ensuring scalability and performance * Selecting appropriate technologies |
| Tester | * Performing various tests * Identifying and documenting bugs * Verifying user experience and game functionality |
| Designer | * Crafting game interfaces * Creating visual assets * Ensuring a cohesive aesthetic across all games |

For effective communication and coordination among team members, we utilize tools like GitHub for task management and project tracking. This platform allows us to assign tasks, set deadlines, and monitor the progress of different project components. Additionally, we use communication tools like Discord for day-to-day conversations and weekly team meetings, ensuring constant and clear communication throughout the project's development.

**Work plan**

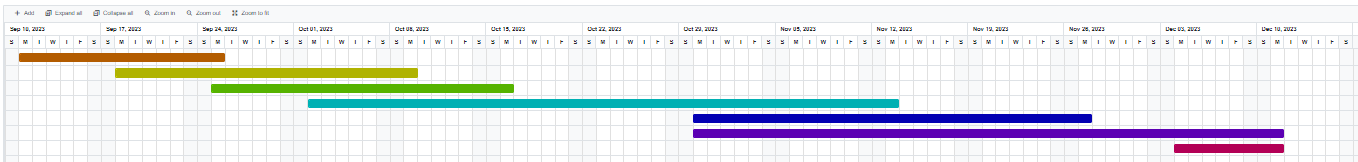
**Chosen Development Process: Pure GitHub**

For the development of this framework we chose to only work with GitHub as it already provides a great platform for posting issues (as features or bugs) and the proceeding with adding features or bugs. We avoided using things like Kanban because it would waste everyones time as this is quite a high-level project and wasting time would not be beneficial to anyone.

**Gantt Chart**

This Gantt chart outlines the project management plan for the Omni-Grid system, structured into seven key phases over three months. It begins with Project Initiation, setting the foundation from September 11 to September 25. This is followed by Requirement Analysis and Design Phase, each spanning roughly two weeks. The Development Phase, the most extended period, takes about a month, emphasizing the bulk of the work. Testing and Documentation run concurrently, reflecting a multitasking approach, before finalizing with Deployment in the first week of December.A screenshot of a computer

Description automatically generated



**Risk analysis**

Effective risk management is crucial for the successful completion of our project. Below is a table outlining potential risks and their corresponding management strategies.

**Table 3. Project risks**

|  |  |
| --- | --- |
| **Risks** | **Risk management strategy** |
| A team member falls ill. | Distribute the work among the remaining team members and adjust deadlines if necessary. |
| Technical difficulties with development tools | Have backup tools or platforms ready; ensure regular training on alternative tools. |
| Unforeseen complexities in game development | Allocate additional time for research and problem-solving; consult with external experts if needed. |
| Delays in task completion | Implement more frequent check-ins and adjust the project timeline accordingly. Increase resources for lagging tasks. |
| Inadequate testing leading to bugs | Allocate more time for testing phases; consider adding more team members to the testing process. |
| Loss of project data | Implement a robust data backup and recovery plan; use version control systems like Git. |
| Changes in project requirements | Maintain a flexible development approach; regularly review and adjust project goals with stakeholders. |
| Communication breakdowns within the team | Establish clear communication protocols; schedule regular team meetings and updates. |
| User interface/design not meeting user expectations | Conduct user surveys and beta testing to gather feedback; be ready to iterate on design based on user input. |

# **1. FORMULATING THE TASK**

This section is dedicated to defining the primary tasks and goals of our project, which involves the development of a digital suite of classic games: Tic Tac Toe, Checkers, and Connect Four. We will outline the specific objectives we intend to accomplish, the scope of our work, and the methodologies we plan to employ to achieve these goals.

Key aspects to be covered in this section include:

* **Project Overview:** A brief introduction to the project, highlighting its purpose and the specific games included in the suite.
* **Objective Clarification:** Detailed explanation of what we aim to achieve with this project, including technical, aesthetic, and user experience goals.
* **Task Breakdown:** A comprehensive breakdown of the tasks involved in the project. This will include everything from initial research and analysis of similar systems to the development of game logic, user interface design, and system implementation.
* **Methodology:** An overview of the development methodology chosen for the project (GitHub), explaining why it is suited to our project’s needs.
* **Scope Definition:** Defining the scope of the project to set clear boundaries on what will and will not be included. This helps in focusing our efforts and resources on the key aspects of the project.

By the end of this section, readers should have a clear understanding of what the project entails, the tasks to be completed, and the strategies we will use to achieve our objectives.

## **1.1. Analysis of similar systems**

In this section, we analyze three digital versions of classic games: Tic Tac Toe, Checkers, and Connect Four. Each system is evaluated for its advantages and disadvantages to inform the development of our project.

**Similar System 1: Digital Tic Tac Toe (Neave Interactive)**

* **Advantages:**
  + Simple, retro-style gameplay appealing to nostalgia.
  + Supports both single-player and two-player modes.
* **Disadvantages:**
  + Basic functionality with limited modern features.
  + No advanced difficulty settings for varied player experience.

**Similar System 2: Digital Checkers (Gametable.org)**

* **Advantages:**
  + Offers multiple game modes and difficulty levels.
  + Features beautiful graphics and easy-to-use interface.
* **Disadvantages:**
  + The design might feel outdated for users seeking modern aesthetics.
  + In-game advertisements can be intrusive.

**Similar System 3: Digital Connect Four (Coolmath Games)**

* **Advantages:**
  + Innovative gameplay with a larger 5x5 grid option.
  + Includes strategic tips and tricks for enhanced gameplay.
* **Disadvantages:**
  + Limited multiplayer options.
  + Basic presentation without significant aesthetic appeal.

**Table 4. Comparison of similar systems**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Neave Interactive** | **Gametable.org** | **Coolmath Games** |
| Single Player Mode | + | + | + |
| Multiplayer Mode | + | - | - |
| Difficulty Levels | - | + | - |
| Advanced Features | - | - | + |
| Graphics and Animations | - | + | - |
| Score Tracking | - | + | - |
| Large Grid Option | - | - | + |
| Strategic Tips | - | - | + |

## **1.2 Functional requirements**

In this section, we detail the functional requirements for our suite of digital board games, including Tic Tac Toe, Checkers, and Connect Four. Functional requirements specify what the system should do and encompass all features and functionalities that the software must provide.

**Table 5. Functional requirements of the system**

|  |  |
| --- | --- |
| **ID** | **Functional requirement** |
| FR1 | **Game Initialization:** Ability to start a new game in each of the three games. |
| FR2 | **Player Interaction:** Capability for players to make moves in the game, with the system validating the legality of these moves. |
| FR3 | **Win/Loss Determination:** The system must be able to determine and display the winner or a draw situation based on the game rules. |
| FR4 | **User Interface:** An intuitive and responsive user interface that displays the game board, player turns, and game status. |
| FR5 | **Single player Support:** Provide options for single-player mode. |

## **1.3 Non-functional requirements**

In this section, we outline the non-functional requirements for our suite of digital board games. Non-functional requirements are critical as they define the system's quality attributes and performance characteristics, setting constraints on the design and implementation.

**Table 6. Non-functional system requirements**

|  |  |
| --- | --- |
| **ID** | **Non-functional requirement** |
| NFR1 | **Performance:** The system should be responsive with minimal lag in gameplay, aiming for a response time of under 2 seconds for all actions. |
| NFR2 | **Usability:** The user interface should be intuitive and easy to navigate for users of all ages and skill levels. |
| NFR3 | **Maintainability:** The system should be easy to update and maintain, with clear documentation for future enhancements. |
| NFR4 | **Environmental:** The system should be designed with environmental sustainability in mind, optimizing resource usage and minimizing energy consumption. |
| NFR5 | **Compatibility:** The games should be compatible across at least one platform, e.g., desktop environments. |

# **2. TASK ANALYSIS**

In the "Task Analysis" section of our documentation, we delve into a detailed examination of the individual tasks that form the backbone of our project. This analytical approach aims to break down complex activities into more manageable units, providing a clear pathway from project conception to completion.

Key elements that will be covered in this section include:

* **Task Identification:** A comprehensive listing of all tasks necessary to take our project from initial planning to final deployment. This includes research, design, programming, testing, and more.
* **Task Breakdown:** Each task will be dissected to understand its components and the steps required to complete it. This granular view will help in allocating resources more efficiently.
* **Dependency Analysis:** We'll explore how tasks are interrelated and identify any dependencies that may affect the project timeline or resource allocation.
* **Time Estimation:** For each task, we'll provide an estimated duration based on complexity and priority, which will be critical for Gantt chart creation and timeline management.
* **Resource Allocation:** This will outline the human, technological, and financial resources assigned to each task, ensuring that the necessary support is available to meet project goals.
* **Risk Assessment:** Each task will be evaluated for potential risks, and strategies will be developed to mitigate these risks proactively.

By the end of the Task Analysis section, we will have a clear understanding of each step required to make the project successful and the challenges that we may face along the way.

## **2.1. Use case diagram and description**

This section presents a UML (Unified Modeling Language) use case diagram that illustrates the various ways users (actors) can interact with the Omni-Grid system. The diagram serves as a visual tool to specify the system's functional requirements and to document user interactions.

**Key Components of the Use Case Diagram:**

* **Actors:** The diagram identifies the types of users interacting with the system. For our suite, the actors include 'Guest' and 'Player'.
* **Use Cases:** These represent the specific actions or behaviors that actors can perform with the system. In our case, use cases include 'Register Account', 'Select Game', 'Play Tic Tac Toe', 'Play Connect Four', and 'Play Checkers'.
* **System Boundary:** This encapsulates all the use cases and defines the scope of the system. It is typically represented as a rectangle.
* **Associations:** Lines connecting actors to their respective use cases, indicating which actor can initiate or participate in which use case.
* **Relationships:** These include 'include', 'extend', and generalization connections that detail the dependencies and hierarchies between use cases.

**Use Case Descriptions:**

Each use case within the diagram is accompanied by a detailed description, as shown in the previous tables. These descriptions provide an in-depth look at the functionality from the user's perspective, including the pre-conditions for each action, the inputs required, the outputs or changes resulting from the action, and the post-conditions once the action is completed.

**Purpose of the Use Case Diagram:**

The use case diagram is a crucial part of the software development process as it:

* Provides a high-level understanding of how the system will be used.
* Helps identify and clarify the requirements of the system.
* Serves as a communication tool between stakeholders, including developers, clients, and users.
* Aids in the planning of the project by mapping out the functionalities that need to be developed and tested.

By the end of this section, readers will have a clear visual representation of the system's user interactions and a comprehensive understanding of each function within the game suite.

A diagram of a game

Description automatically generated

**Table 7. Register Account**

|  |  |
| --- | --- |
| **Function** | Register Account |
| **ID** | UC01 |
| **Description** | Allows guests to create a player account in the system |
| **User roles** | Guest (Unregistered player) |
| **Prie-conditions** | Internet connectivity is available |
| **Inputs** | Player details (Username, Name, Surname, Password) |
| **Outputs** | Confirmation email, User ID |
| **Action** | 1. Guest selects 'Register' option. 2. System displays registration form. 3. Guest fills in personal details and submits the form. 4. System checks for existing user in the data store. 5. System creates a new player account and data store entry. |
| **Post-conditions** | Guest becomes a registered Player and can log in |
| **Side effects** | None. |

**Table 8. Select Game**

|  |  |
| --- | --- |
| **Function** | Select Game |
| **ID** | UC02 |
| **Description** | Allows players to choose which game they want to play from the available selection. |
| **User roles** | Registered Player |
| **Prie-conditions** | The player is logged into the system. |
| **Inputs** | The player's choice of the game. |
| **Outputs** | The selected game interface is loaded. |
| **Action** | 1. Player logs into the system. 2. The system presents the available games. 3. Player selects a game. 4. The system loads the game interface. |
| **Post-conditions** | The player is ready to start playing the selected game. |
| **Side effects** | None. |

**Table 9. Play Tic Tac Tor**

|  |  |
| --- | --- |
| **Function** | Play Tic Tac Toe |
| **ID** | UC03 |
| **Description** | Engage the player in a game of Tic Tac Toe. |
| **User roles** | Registered Player |
| **Prie-conditions** | The player has selected Tic Tac Toe from the game options. |
| **Inputs** | Player's moves during the game. |
| **Outputs** | The updated game board after each move, result of the game (win/lose/draw). |
| **Action** | 1. Player selects Tic Tac Toe from the game options. 2. The system initializes a new Tic Tac Toe game. 3. Player makes a move. 4. The system updates the game board and checks for a win or draw. 5. Repeat steps 3-4 until the game ends. |
| **Post-conditions** | The game ends with a win, loss, or draw. |
| **Side effects** | None. |

**Table 10. Play Connect Four**

|  |  |
| --- | --- |
| **Function** | Play Connect Four |
| **ID** | UC04 |
| **Description** | Allows players to engage in a game of Connect Four, aiming to connect four of their own discs in a row. |
| **User roles** | Registered Player |
| **Prie-conditions** | Player has chosen Connect Four from the game selection. |
| **Inputs** | Player's selection of columns to drop their discs. |
| **Outputs** | Updated board state with the player's and opponent's discs. |
| **Action** | 1. The system presents a Connect Four grid to the player. 2. Players take turns dropping discs into the columns. 3. The system updates the board after each move and checks for a win condition. 4. The game continues until a player achieves four in a row or the grid is full. |
| **Post-conditions** | The game concludes with a winner or a draw. |
| **Side effects** | None. |

**Table 11. Play Checkers**

|  |  |
| --- | --- |
| **Function** | Play Checkers |
| **ID** | UC05 |
| **Description** | Engages players in a traditional game of Checkers with standard rules. |
| **User roles** | Registered Player |
| **Prie-conditions** | Player has selected Checkers from the list of games. |
| **Inputs** | Player’s choices of piece movements. |
| **Outputs** | The updated Checkerboard after each move, and the game outcome. |
| **Action** | 1. Player selects Checkers from the game options. 2. The system sets up a new Checkerboard. 3. Players alternate moves according to the rules of Checkers. 4. The system updates the board and checks for kinged pieces or wins. 5. The game continues until one player wins or no legal moves are left. |
| **Post-conditions** | The game ends with a winner or in a stalemate situation. |
| **Side effects** | None. |

## **2.2 Activity diagram(s) and description (s)**

**Activity Diagram 1: Player Registration Process**

**Description:** This diagram outlines the process a guest goes through to register and become a player. It starts with a check to determine if the user is a guest. If confirmed, the system prompts the user with a registration form. Once the guest enters their details, the system validates this information. If the details are valid, the system creates a new player account and sends a confirmation email. Should the details be invalid, an error message is displayed, and the process ends.

A diagram of a flowchart

Description automatically generated

**Activity Diagram 2: Game Selection Process**

**Description:** This diagram represents the steps a player takes to select a game to play. After logging in, the player is presented with a list of available games. The player makes a selection, prompting the system to load the appropriate game interface, signifying the end of the game selection process and the beginning of gameplay.

A diagram of a game

Description automatically generated

**Activity Diagram 3: Tic Tac Toe Gameplay**

**Description:** The gameplay process for Tic Tac Toe is depicted in this diagram. It begins with the initialization of the game and progresses as the player makes a move. The system then checks for a win or draw condition. If neither condition is met, the turn switches to the opponent. This cycle continues until a win or draw condition is achieved, at which point the game displays the result and ends.

A diagram of a game

Description automatically generated

**Activity Diagram 4: Connect Four Gameplay**

**Description:** In the Connect Four gameplay diagram, the game starts and the player chooses a column to drop their disc. The system then checks if there's a sequence of four connected discs, indicating a win. If a player wins, the result is displayed and the game concludes. If not, the turn switches to the opponent, and play continues until there's a winner.

A diagram of a game

Description automatically generated

**Activity Diagram 5: Checkers Gameplay**

**Description:** This activity diagram illustrates the process of playing a game of Checkers. After game initialization, players take turns moving their pieces. The system checks for opportunities to jump the opponent's pieces. If a jump is possible, it's carried out, and the opponent's piece is removed. The system also checks for a win condition after every move. If a player wins, the winner is displayed, and the game ends. If the game continues, the turn switches to the opponent.

A diagram of a game

Description automatically generated

## **2.3 The entity-relationship diagram and its description**

A diagram of a diagram

Description automatically generated

**User Entity:**

The User entity represents the registered players of the game suite. Each user is uniquely identified by a primary key UserId. The entity stores the user's credentials (UserName and UserPassword) and personal information (UserFirstName and UserLastName). This information is crucial for user identification, authentication, and personalization of the game experience.

**Attributes:**

* UserId: The unique identifier for each user.
* UserName: The chosen username for the player to log in.
* UserPassword: Encrypted password for account security.
* UserFirstName: The first name of the user.
* UserLastName: The last name of the user.

**Score Entity:**

The Score entity keeps track of the game outcomes and performance statistics for each player. It is identified by a ScoreId and linked to the User entity through UserId, establishing a relationship between users and their gaming records.

**Attributes:**

* ScoreId: The unique identifier for each score entry.
* UserId: A foreign key that establishes a link to the corresponding user.
* Wins: The total number of games won by the user.
* Losses: The total number of games lost by the user.
* WinRate: A calculated field representing the win-to-loss ratio.

**Relationships:**

* User has Scores: A one-to-many (1:M) relationship. Each user can have multiple score records, reflecting their participation in various games within the suite. However, each individual score record is associated with one, and only one, user.

**Cardinality:**

* The cardinality of the relationship between User and Score is one-to-many, indicated by the notation (1:M). This means that while each score record specifically pertains to a single user, each user can have numerous score records associated with their account.

**ER Diagram Structure:**

In the ER diagram, User is depicted as a rectangle labeled with its attributes. Similarly, Score is depicted as another rectangle with its attributes. A line connects User to Score, and the end near Score fans out to represent the "many" part of the one-to-many relationship. An arrow or a key symbol might be used to indicate the foreign key reference from Score to User.

This ER diagram and its descriptions provide a clear overview of how player data and game performance metrics are structured within the data store. It facilitates an understanding of how different data points are interconnected, which is essential for data store management, querying, and reporting `within the game suite system.

## **2.4 Layout of the user interface**

This section provides an overview of the user interface (UI) layout for the Classic Games Suite. The UI is designed to be intuitive and user-friendly, ensuring that players can navigate through the system with ease.

**User Table:**

* A detailed view of the user accounts, including login credentials and player profiles.
* It's directly connected to the login and registration forms, ensuring seamless user data flow.

**Score Table:**

* Displays player scores and rankings.
* Interconnected with the game modules to update scores in real-time post-gameplay.

**Welcome Screen:**

* The initial interface a player encounters upon starting the application.
* Offers navigation options to different sections of the game suite such as game selection, user account management, and leaderboards.

**Register Form:**

* A dedicated interface for new users to create their accounts.
* Simple fields to enter necessary information such as username, email, and password.
* Directly linked to the User Table in the data store for account creation and verification.

**Login Form:**

* Allows returning users to access their accounts.
* Contains fields for username and password with validation checks for secure entry.
* Provides a pathway to the Welcome Screen upon successful authentication.

**Game Selection Interface:**

* Lets the player choose from the available games to play.
* Presents a list or gallery of games, with options to select and load a game for play.

**Game Interfaces:**

* Once a game is selected, the relevant game interface is loaded.
* It's tailored to each game's specific requirements, with controls and displays appropriate for Tic Tac Toe, Connect Four, or Checkers.

This layout ensures a coherent flow from user onboarding to game selection and play. The interface design prioritizes accessibility and engagement, with clear visual cues and responsive design elements to enhance the overall user experience. Each component of the UI is strategically placed to provide a seamless transition between different user actions and system responses.

A diagram of a website

Description automatically generated

# **3. SOFTWARE IMPLEMENTATION**

**Overview:**

The "Software Implementation" section details the practical realization of the game suite's software components. It will delineate the structure of the codebase, explaining the roles of various files, classes, and their methods. This section is designed to provide a comprehensive look into the technical underpinnings of the system, offering insights into the design decisions and the way software functions to fulfill the requirements set out in the design phase.

**Content:**

* **File Structure:** A walkthrough of the software's file hierarchy, highlighting the purpose of each file within the project. This includes source code files, configuration files, and resource files that are part of the application's ecosystem.
* **Class Descriptions:** For each class within the codebase, this subsection will outline the class's responsibility, its relationship with other classes, and its place within the overall architecture. Key properties and methods will be described, with an emphasis on their actions, inputs, and expected outputs.
* **Methods Functionality:** Detailed descriptions of the critical methods, including their parameters, process flow, and the structure of the results they produce. This will encapsulate the logic behind core functionalities such as user authentication, game state management, score tracking, and data persistence.
* **Data Store Physical Model:** If applicable, the physical model of the data store will be presented. This includes tables, columns, data types, indexes, constraints, and relationships between tables. It serves as a blueprint for understanding how data is stored, organized, and accessed.
* **Data Store Description:** A narrative describing the data store role in the system, explaining how it supports the application's features, the rationale behind its schema design, and how it ensures data integrity and efficiency.
* **Software Constructs:** Explanation of other significant constructs like components, modules, APIs, and libraries. The interrelationships between these elements and how they work together to form a cohesive system will be detailed.
* **Code Standards:** The section will address the coding standards followed throughout the development process, including naming conventions, commenting strategies, and code organization practices.

**Presentation:**

* All code snippets included for illustration will be presented in the Courier New font to maintain clarity and consistency.
* Instead of screenshots, code will be embedded directly into the documentation to allow for easy text processing and searching within the BD inventory.

**Purpose:**

This section aims to convey a deep understanding of the software's inner workings to developers, maintainers, and technical auditors. It serves as both a descriptive and instructional piece, ensuring that the software can be efficiently maintained, upgraded, or integrated with other systems in the future.

## **3.1. System architecture and description**

**Users Component:**

The architecture begins with the Users, the cornerstone of the system. They are the players who interact with the application, providing input through the Front-end UI and receiving feedback in return. The flow from Users to Front-end UI represents this interaction, where they can select games to play, input moves, and navigate through various game options.

**Front-end UI Component:**

The Front-end UI is the visual interface of the application. It is composed of game screens where the gameplay takes place and menus for navigation and game selection. The Front-end UI is designed to be intuitive, responsive, and engaging, offering an immersive gaming experience. It collects data from the Users, such as game moves or choices, and displays results like game outcomes or leaderboards.

**Game Logic (Back-end Logic) Component:**

The Game Logic is the back-end part of the system, where the core application logic resides. It processes the data collected from the Front-end UI, such as game moves, and determines the outcomes based on the rules of the games. The Game Logic is responsible for maintaining the state of the game, enforcing game rules, calculating scores, and managing the game progression. It sends updates back to the Front-end UI to reflect changes in the game state.

**Data Store System Component:**

The Data Store System is a critical component for data persistence. It stores all the game states, which includes the current status of each game session, and user profiles, containing player information and statistics. The Data Store System interacts with the Game Logic, storing information after game moves are processed and outcomes are determined. It also retrieves data as needed to maintain continuity of gameplay and to provide a personalized experience for each user.

**Overall Flow:**

The diagram shows a clear flow of interactions:

* Users interact with the Front-end UI.
* The Front-end UI communicates with the Game Logic to process those interactions.
* The Game Logic, in turn, interacts with the Data Store System to store and retrieve game-related data.
* Additionally, there is a bidirectional flow between the Front-end UI, the Game Logic and DBS indicating that the game state is continuously updated on the user's screen based on the logic processed on the server-side.

This system architecture ensures a robust, maintainable, and scalable gaming platform, enabling a seamless experience from user input to game state management and persistence.

A diagram of a computer

Description automatically generated

## **3.2. System files and description**

* **Source Folder**: Contains the main application code, including game logic, user interface handling, and system integration scripts.
* **Tests Folder**: Includes unit and integration test files that are designed to validate the correctness and stability of the application code.
* **Documentation Folder**: Houses all the documentation related to the project, such as setup instructions, design documents, and usage guidelines.
* **Library Folder**: Likely stores third-party libraries or dependencies required by the application.
* **Examples Folder**: May contain sample configurations, demos, or additional scripts showcasing the application's capabilities.
* **.github Folder**: Typically holds the configuration files for GitHub actions or workflows for CI/CD processes.
* **.vscode Folder**: Contains settings for the Visual Studio Code environment, such as debug configurations and recommended extensions.

## **3.3. Description of classes and their methods**

The UserMenu class in the Omni-Grid project manages user interactions with the menu, including pre and post-login options, user registration, and data store initialization. The Checkers class implements the game logic for Checkers, handling game moves, win and draw conditions, and player setup. The ConnectFour class, similar to Checkers, manages the Connect Four game, including moves, win conditions, and player interactions. The TicTacToe class encapsulates the Tic Tac Toe game logic, handling moves, checking for win or draw conditions, and updating the game grid. Each class is tailored to its specific functionality within the game suite, ensuring a smooth gaming experience.

**Table 12. UserMenu Class**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Attribute \*m\_Database | Reference to the data store for user data persistence. |
| Attribute \*m\_User | Reference to the current user of the menu. |
| Method UserMenu() | Constructor to initialize the user menu. |
| Method ~UserMenu() | Destructor to clean up the user menu. |
| Method Start() | Begins the user menu process. |
| Method FirstRun() | Checks and sets up initial data store state. |
| Method PreLoginMenu() | Displays options available before user login. |
| Method Login() | Manages the user login process. |
| Method Register() | Handles new user registration. |
| Method PostLoginMenu() | Displays options available after user login. |

**Table 13. Checkers Class**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Attribute m\_Supers | Stores references to super pieces in the game. |
| Attribute m\_SelectedPiece | Reference to the currently selected piece. |
| Attribute m\_Pieces | Keeps track of piece positions. |
| Method Checkers() | Constructor for Checkers game initialization. |
| Method ~Checkers() | Destructor to clean up the user menu. |
| Method TryMakeMove() | Attempts to make a move in the game. |
| Method IsWinningCondition() | Checks for a win condition. |
| Method IsDrawCondition() | Checks for a draw condition. |
| Method SetupPlayers() | Sets up the players in the game. |
| Method Initialize() | Initializes the game with a user. |
| Method OnGUIUpdateGrid() | Updates the game grid on the GUI. |
| Method OnGUIUpdateGridHover() | Updates GUI when hovering over the grid. |

**Table 14. ConnectFour Class**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Attribute alpha | Used for the transparency of circles in the UI. |
| Attribute alphaSpeed | Controls the speed of UI transitions. |
| Method ConnectFour() | Constructor to initialize Connect Four game. |
| Method ~ConnectFour() | Destructor to clean up Connect Four game. |
| Method TryMakeMove() | Attempts to make a move in the game. |
| Method IsWinningCondition() | Checks for a win condition. |
| Method IsDrawCondition() | Checks for a draw condition. |
| Method SetupPlayers() | Sets up the players in the game. |
| Method Initialize() | Initializes the game with a user. |
| Method OnGUIUpdateGrid() | Updates the game grid on the GUI. |
| Method OnGUIUpdateGridHover() | Updates GUI when hovering over the grid. |

**Table 15. TicTacToe Class**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Method TicTacToe() | Constructor to create Tic Tac Toe game instance. |
| Method ~TicTacToe() | Destructor to clean up Tic Tac Toe game instance. |
| Method TryMakeMove() | Attempts to make a move in the game. |
| Method IsWinningCondition() | Checks for a win condition. |
| Method IsDrawCondition() | Checks for a draw condition. |
| Method SetupPlayers() | Sets up the players in the game. |
| Method Initialize() | Initializes the game with a user. |
| Method OnGUIUpdateGrid() | Updates the game grid on the GUI. |
| Method OnGUIUpdateGridHover() | Updates GUI when hovering over the grid. |
| Method DrawX() | Draws an 'X' mark at the specified board position. |
| Method DrawO() | Draws an 'O' mark at the specified board position. |

# **4. INSTALLATION AND USER INSTRUCTIONS**

This section provides detailed guidance on installing and using the Omni-Grid system. It covers software dependencies, computer hardware requirements, deployment steps, system launch procedures, removal instructions, and user guidelines. The focus is on ensuring users can effectively set up, operate, and manage the system, addressing both software and hardware aspects vital for optimal performance and functionality.

## **4.1. Dependencies on other software products**

Omni-Grid depends on various software products like Git, CMake, Strawberry Perl, Python, MSVC v143 compiler, optional vcpkg, and Doxygen. Proper setup, including environmental path configuration and selection of correct installation options, is crucial for the system's functionality.

## **4.2 Computer hardware parameters**

While specific hardware requirements aren't detailed, the system likely needs a modern processor, adequate RAM (8GB or more recommended), a capable graphics card for rendering games, and sufficient hard drive space for installation and operation.

## **4.3 Deployment Schedule**

The deployment process would involve a structured rollout of the system components, adhering to the software dependencies and ensuring proper configuration of each component.

## **4.4. Launch description**

Launching the system involves initializing the development environment, including SSH key generation for GitHub, submodule updates, and script execution for project setup, build, and installation.

## **4.5 Removal steps**

To remove the system, users would uninstall the software, reverse environment path changes, and remove any residual files or dependencies associated with the software.

## **4.6 Instructions for use**

Detailed instructions cover the complete setup and usage of the Omni-Grid system, from environment setup, handling of submodules, to resolving potential issues like missing .dll files.

# **CONCLUSIONS AND PROPOSALS**

**Overview:**

This section summarizes the outcomes of the Omni-Grid project, comparing the results with initial goals and expectations. It scrutinizes the project's alignment with its defined objectives and highlights any deviations from the anticipated results.

**Conclusions:**

1. **Analysis of Similar Systems:** Insights gained from studying comparable gaming platforms, highlighting Omni-Grid's distinct features or opportunities for improvement.
2. **Technical Challenges:** Discussion of any technical obstacles encountered during development and strategies employed to overcome them.
3. **Objective Fulfillment:** Evaluation of how well the project met its stated objectives and the implications of any unmet goals.
4. **Future Enhancements:** Identifying potential areas for further development or upgrades to enhance the system's capabilities.

**Individual Contributions:**

* **Vitold Skuder:** Led the documentation efforts for the Omni-Grid project. Authored a comprehensive Word document encompassing all aspects of the project, from system architecture to user instructions. I meticulously crafted and incorporated various charts and diagrams, ensuring clarity and coherence in the documentation. Additionally, played a pivotal role in devising the project plan, outlining the strategies, and setting the course for the project's successful execution. My contributions were instrumental in providing a structured and detailed narrative of the entire development process.
* **Nazar Lavkart:** Abstracted the initial code. Utilized the framework to create both the ConnectFour game and the Checkers game. Debugged and bugfixed several code segments relating to the framework. Created tests for ConnectFour and Checkers. Wrote UAT documents for ConnectFour and Checkers.
* **Sebastian Termen:** Architecturized the initial layout of the framework. Created the first prototypes. Abstracted the code as was needed. Documented all the code. Created tests for the core of the framework as well as the Tic Tac Toe game. Managed the GitHub repository. Created a tool for building and maintaining the project (CPM). Expanded the Durlib library if functionality was missing.

This segment is critical for understanding the project's impact and its potential evolution, offering a foundation for future advancements.

# **LIST OF SOURCES OF INFORMATION**

* <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>
* <https://www.lucidchart.com/pages/examples/mockup-generator>
* <https://www.dafontfree.io/courier-new-font/>
* <https://app.diagrams.net/>
* <https://www.lucidchart.com/pages/uml-use-case-diagram>
* <https://www.lucidchart.com/pages/er-diagrams>