

EECE 4520 Reversi Final Report, Group 2

Christian Bender, Luke Wisner, Owen Zhang
2023-04-24

Overview

Our goal is to develop an online Reversi program that allows users to play the game against each other over the internet. The program should be user-friendly and accessible, with an intuitive interface and clear instructions for gameplay. The program should also be secure and reliable, ensuring that users' data and gameplay progress is protected. The project will be developed from the ground up, starting from zero.

This final summary report provides an overview of the project to develop an online Reversi program. We planned to accomplish this by going through the proper steps of creating a larger scale project, using UML to design and diagram our project before jumping into the code. In this document we have compiled each step of the project.

Requirements Analysis

1: Introduction

1.1: Purpose of the system

The purpose of this system is to provide an educational experience for students at Northeastern University. If development goes properly, the system may also serve as a competitor to current online Reversi games. This project is being developed for EECE4520 (Software Engineering) at Northeastern University.

1.2: Scope of the System

The system will allow users to:

1. Create an account and log in
2. Implementation user authentication and registration system with a database
3. Provide a user interface (including the game board, game pieces, and controls)
4. Provide matchmaking functionality to allow users to play against each other
 - a. Skill-based matchmaking with ELO system
 - b. Invite players for friendly games
 - c. Play locally against each other on the same device
5. AI-powered single-player gameplay
6. Provide functionality for users to save games in progress and resume them later.
7. Provide functionality for users to customize the Reversi board and game pieces, including color schemes and themes.

1.3: Objectives and Success Criteria

The system should have a front end GUI that can support multiple users, and a back end that is structured using MVC architecture. The system should be able to host multiple local multiplayer or AI games for different users at the same time, as well as host online games for users and track wins, losses, ELO, and top players using a database. Both guest and logged-in user accounts should be supported. Success will be tested individually for each game mode,

with both user and automated testing being employed. If the system has all aforementioned gamemodes with each operating as expected, this will be treated as a success.

1.4: Definitions, Acronyms

GUI: Graphical User Interface

ELO: Method for calculating the relative skill of players in zero-sum games

MVC: Model - View - Controller architecture

1.5: References

Node.js: <https://nodejs.org/en/docs>

Socket.IO: <https://socket.io/docs/v4/>

MySQL: <https://dev.mysql.com/doc/>

1.6: Overview

The system being developed is a Reversi game which will provide a GUI so that the user can interact with it. This development is being done for Northeastern University for student education. The system will be designed to support 3 primary game modes: local multiplayer, play against AI, and online multiplayer. The system will also support account creation, login, and guest play. For online play, the system will track games won, games lost, and ELO.

2: Current System

There is no current system in place. The game will be developed entirely from scratch using JavaScript, node, and npm libraries.

3: Proposed System

3.1: Overview

The system being developed is a Reversi game which will provide a GUI so that the user can interact with it. This development is being done for Northeastern University for student education. The system will be designed to support 3 primary game modes: local multiplayer, play against AI, and online multiplayer. The system will also support account creation, login, and guest play. For online play, the system will track games won, games lost, and ELO.

3.2: Functional Requirements:

1. GUI
2. Player login with username and password
3. Player signup with username and password
4. Login as guest
5. Game mode selection
6. Play against another player locally
7. Play against a computer

- a. User control of AI level (controls the depth of computations)
8. Play against another player online
 - a. System should display all online users
 - b. System should let each player decide who to play against
9. Track the number of wins and losses for each player and compute ELO rating
10. Leaderboard which displays the top 5 highest scoring players and their ratings

3.3: Non-functional Requirements

1. Accessible on multiple devices and browsers.
2. Fast response time and minimal latency during gameplay.
3. High level of availability and uptime, with minimal downtime for maintenance.
4. Scalable, allowing for an increasing number of users and matches.
5. Secure user authentication and data encryption to protect users' personal information and gameplay progress.
6. Clean and simple user interface with easy navigation.
7. Display clear instructions for gameplay and user actions.
8. Provide users with a visual representation of the Reversi board and game pieces.
9. Provide users with real-time updates on gameplay progress, including moves and score updates

3.4: System Models

3.4.1: Scenarios

We created a diagram and description of what we would like a user to experience when using our program, to help guide us through the boundaries and functionality needed for the program.

<i>Scenario name</i>	playReversi
<i>Participating actor instances</i>	alice:Player, joe:Player, system:System
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. Alice opens the game 2. Alice logs in with a username and password 3. System recognizes the username and password, Alice proceeds to the game menu 4. Alice is presented with a menu to play vs AI, play online, play locally, quit the game, logout, or customize the game settings 5. Alice chooses to play online 6. Alice is matched with another player, Joe 7. Alice and Joe finish their game of reversi, and the system determines Alice as the winner 8. Alice gains ELO and Joe loses ELO according to each of their ratings 9. Alice is done playing, and quits the game
<i>Entry conditions</i>	<ul style="list-style-type: none"> • The player is logged into the game
<i>Exit conditions</i>	<ul style="list-style-type: none"> • The player quits the game

Play Reversi Scenario

3.4.2: Models

For the Object Model, we created class diagrams which we updated throughout the project, featured in the Class Diagrams section of this document. This diagram represents the objects involved and their relationships to each other.

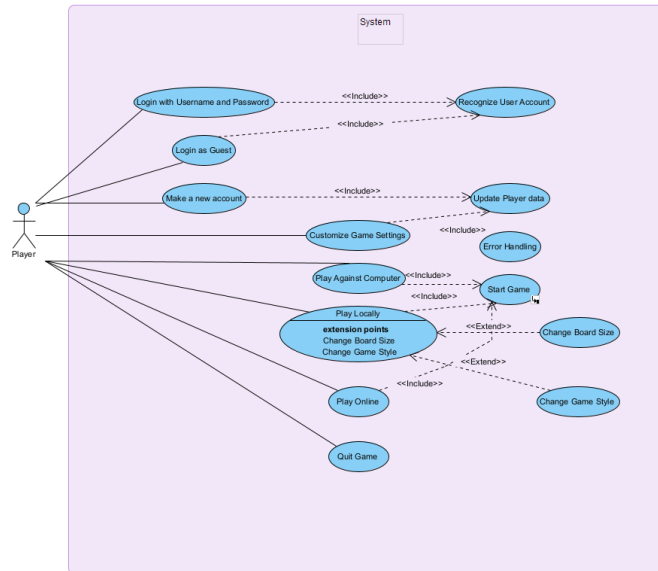
For the Dynamic Model, we made various use case diagrams and descriptions detailed in the Use Cases section. These diagrams describe various scenarios which are expected to happen, and detail the logic following each case.

User Stories

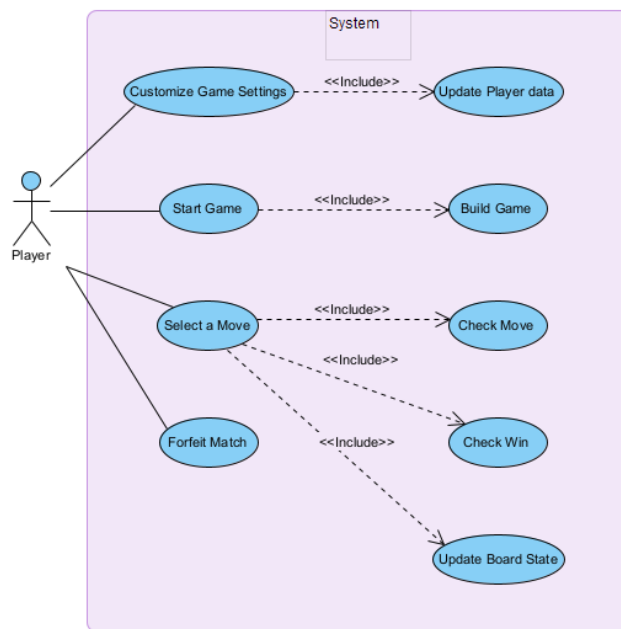
1. As a player, I want to be able to create an account, so that I can save my progress and track my gameplay history.
2. As a player, I want to be able to search for opponents based on skill level or availability, so that I can find challenging matches or play with friends.
3. As a player, I want to be able to invite other players to play Reversi matches, so that I can compete against specific opponents.
4. As a player, I want to be able to view my opponent's gameplay history and win/loss record, so that I can assess their skill level and adjust my strategy accordingly.
5. As a player, I want to be able to communicate with my opponent during gameplay via a chat feature, so that we can discuss our moves or provide feedback.
6. As a player, I want to be able to save a game in progress and resume it later, so that I can take breaks or play over multiple sessions.
7. As a player, I want to be able to view replays of completed matches, so that I can analyze my gameplay and improve my strategy.
8. As a player, I want to be able to customize the Reversi board and game pieces, so that I can personalize my playing experience.

Use Cases

We created multiple Use Case diagrams and Case Descriptions to help detail the process for specific situations. These diagrams showcase how the user will interact with the system, including the actors and their interactions with the program. We used these diagrams to help outline the boundaries and scope of the whole program.



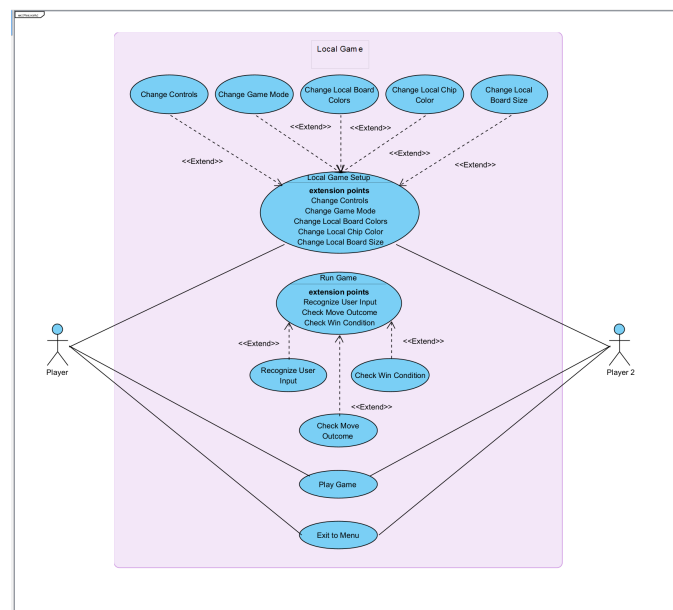
Main Use Case Diagram



Start Game Use Case Diagram

<i>Use case name</i>	startGame
<i>Participating actors</i>	Initiated by Player Communicates with System
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. The player initiates start game 2. The system builds the game 3. The player makes a move 4. The system checks if the move is legal 5. The system updates the board if legal 6. The system checks the board to see if a player has won, and if so, determines the winner 7. The system terminates the game
<i>Entry conditions</i>	<ul style="list-style-type: none"> • The player has initiated the game
<i>Exit conditions</i>	<ul style="list-style-type: none"> • The system has determined a winner

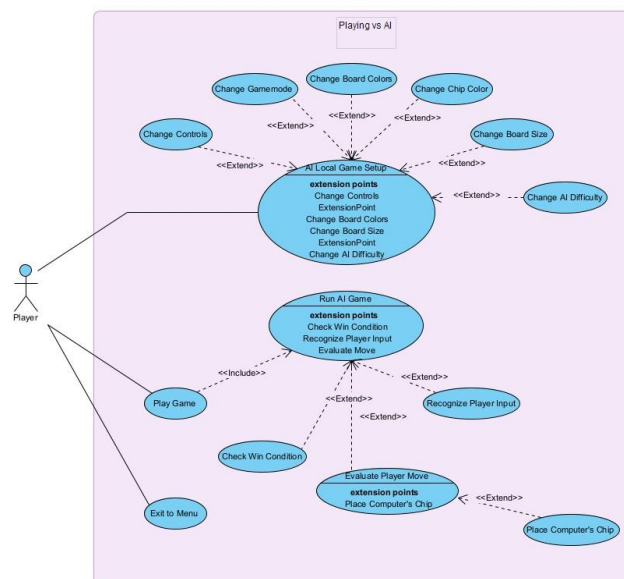
Start Game Use Case Description



Play Locally Use Case Diagram

<i>Use case name</i>	PlayLocally
<i>Participating actors</i>	Communicates with Player Communicates with Player 2 Communicates with System
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. Player 1 selects Play Locally 2. Player 1 or 2 may change their chip colors, board color, board size, and game mode. Otherwise the local game setup loads default settings. 3. Player 1 starts the game once all of the game settings have been adjusted to satisfaction. 4. The system runs the game, and both players play the game using the same keyboard / mouse. The game recognizes user inputs, checks move outcomes, and checks win conditions. 5. The game finishes with one player winning, or one player Exits to Menu before the game is over.
<i>Entry conditions</i>	<ul style="list-style-type: none"> • Player 1 is logged in with account or as guest • Player 1 has selected Play Locally
<i>Exit conditions</i>	<ul style="list-style-type: none"> • Player 1 or 2 wins the local game • Player 1 exits to menu

Play Locally Case Description



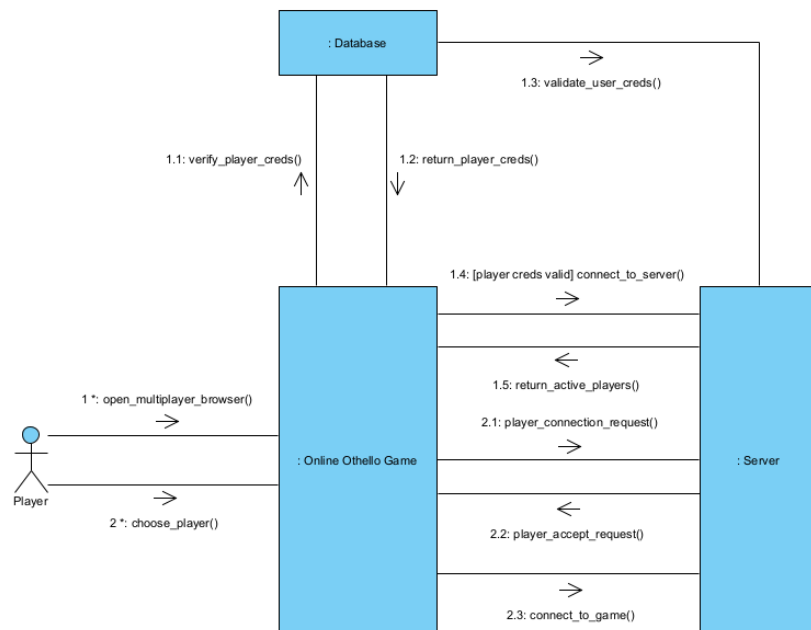
Play vs Ai Use Case Diagram

<i>Use case name</i>	PlayVsAI
<i>Participating actors</i>	Initiated by Player Communicates with System
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. Player selects Play Vs AI gamemode 2. Player may choose to change any of the game settings like the gamemode, controls, board size, board color, chip color, or AI difficulty 3. Player begins playing which runs the AI game 4. The system runs the game, which involves checking if the player or AI has won the game, recognizing if the player's move is valid, and evaluating the player's move 5. Evaluating the player's move means updating the board to reflect the added chip, but it also sends the new board state through our AI reversi algorithm to place the computer's next chip 6. Play continues in this way until either the player or the AI has won or the player has exited the game
<i>Entry conditions</i>	<ul style="list-style-type: none"> • Player is logged in with an account or as guest • Player has selected Play Vs AI
<i>Exit conditions</i>	<ul style="list-style-type: none"> • Player or AI win the game • Player exits the game

Play vs AI Case Description

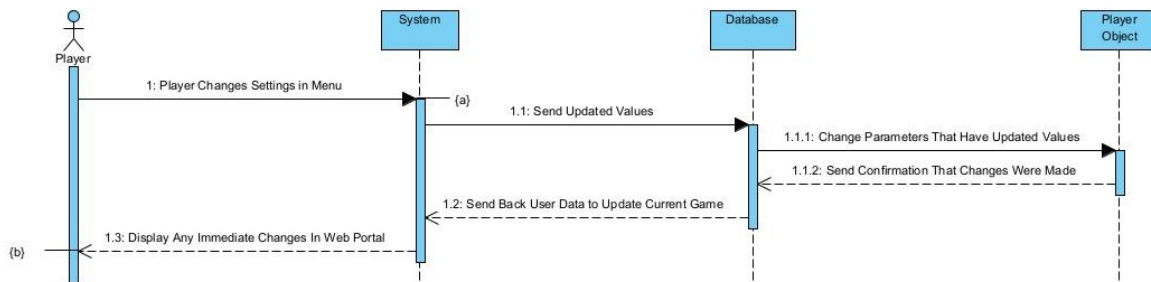
Sequence and Communication Diagrams

The primary communication between objects happened between the view server and database. It was therefore logical to create a communication diagram for online matchmaking, as this gamemode had the most connections between the server and view. The online matchmaking communication diagram is shown below.



Online Matchmaking Communication Diagram

A sequence diagram was created to highlight in more detail how the logging in process should work. This is shown in the diagram below.



Update User Information Sequence Diagram

Software Architecture and Subsystem Decomposition

See section 10 below for the system component diagram which visually highlights the summary of the subsystem decomposition description below.

View:

- View runs client-side to handle login, sign up, select mode, and make move
- These functions all interact with the GUI and send their results to the server

- If user credentials are sent to the server, it authenticates it through the database information
- User sessions are also pulled from the database if the user has a previous game they can resume
- All other server interactions happen through the controller that manages board settings, checks if valid move, and makes a move

Server:

- The server contains the connection to the database and the controller and model which handle the game logic
- When passed user credentials, the server can either validate the user in the database and log them in or create a new user
- The controller can change the board settings of the game, and handle moves made through the GUI
- Handling a move requires checking if it is valid, and then if it is the board must be updated
- Updating the board and the current game happens through the connected model

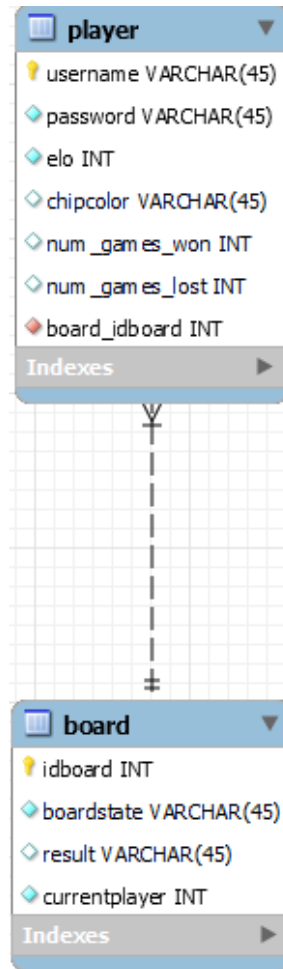
Database:

- The database contains user information that is pulled by the server to validate logins
- The database can also add new users when the server sends it new credentials
- Along with the user information, previous games are also stored and can be called by the server if the player wants to resume them

System and Software Design

The program is structured using MVC architecture. The model features the game state, rules, and algorithms for calculating moves. The Model is independent of the View and Controller and provides methods for updating and retrieving data. The View displays the game board, pieces, and game information. It provides user input methods such as clicking on the board to make a move. The Controller receives input from the View, updates the Model accordingly, and then notifies the View of any changes. It handles game events such as player turns and the end of the game.

The network is designed with a client-server database, using Node.js, Socket.IO and MySQL. The server side is built using Node.js and utilizes Socket.IO and MySQL modules for real time, bidirectional communication, and database management to store game and user data. The database is designed with the following ERD:



Database ERD Diagram

On the client side, we used HTML, JavaScript, and CSS, along with the Socket.IO module, which work together to render the board, pieces, UI, etc. and send the user input to the server.

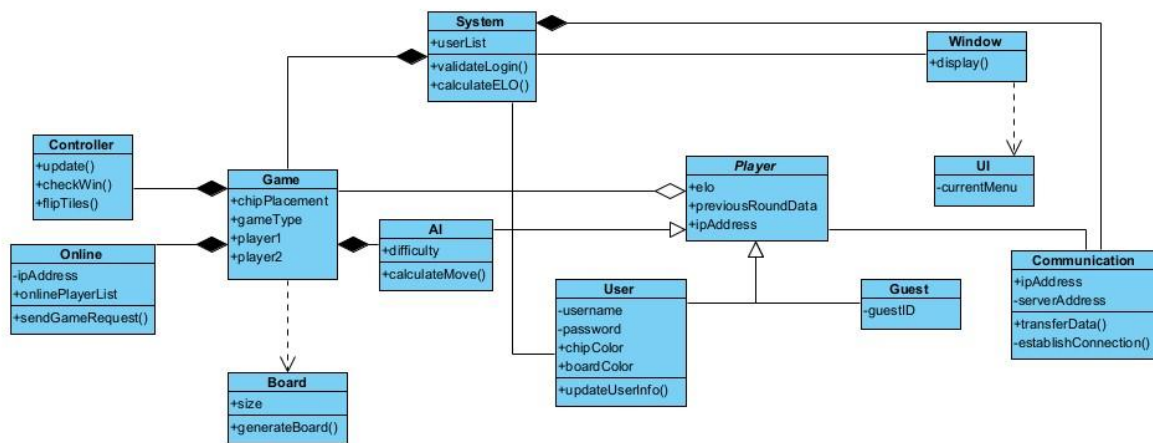
The general workflow proceeds as followed:

- The player logs in to the system and searches for a game.
- The server checks if there are any other players searching. If not, the player waits for other players to join.
- Once two players are in the same game room, the server creates a new game and initializes the game state.
- The server sends the game state to both players, and the game begins.
- Each player makes their moves by clicking on the board, and the client sends the move to the server.
- The server updates the game state in the database, validates the move, and sends the updated game state to both players.

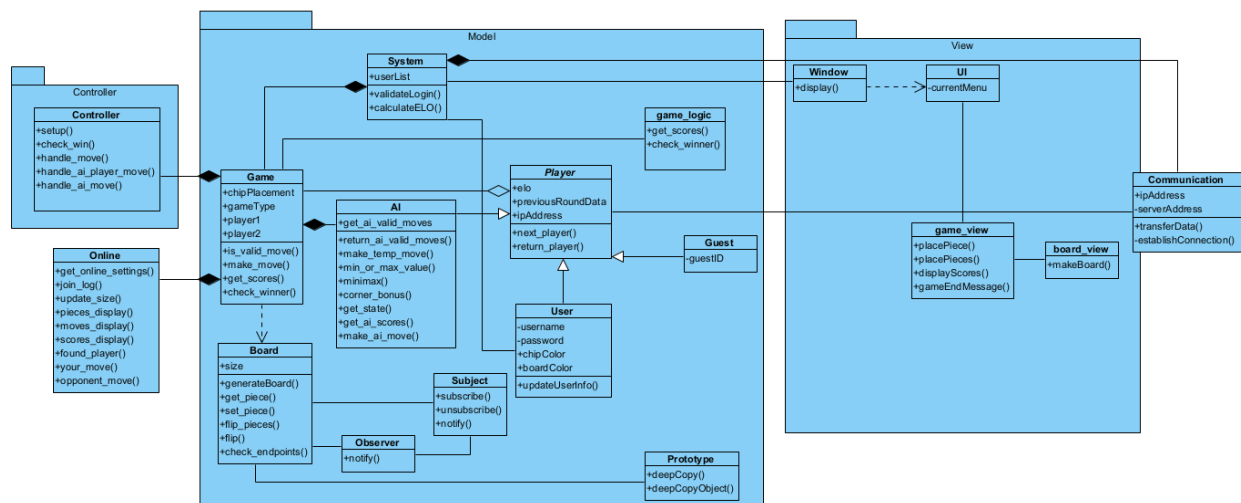
- g. Steps e and f are repeated until the game is over, either because the board is full or there are no more valid moves.
- h. The server updates the game status in the database, and sends the game results to both players.

Class Diagrams

One parent class diagram was used to represent the reversi system over the course of the semester. This class diagram was updated multiple times to represent the ever changing system. Both the first draft class diagram from Milestone 1 and the final draft class diagram from Milestone 2 are shown below.



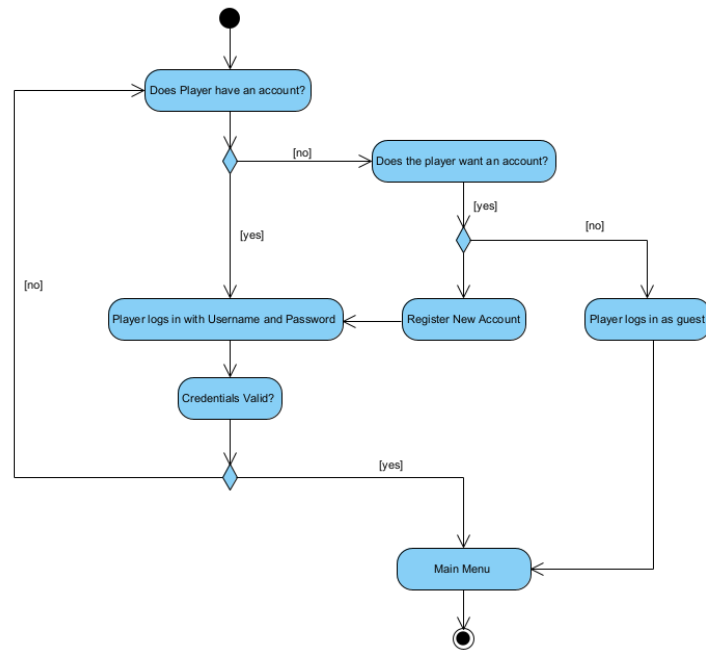
First Draft Class Diagram



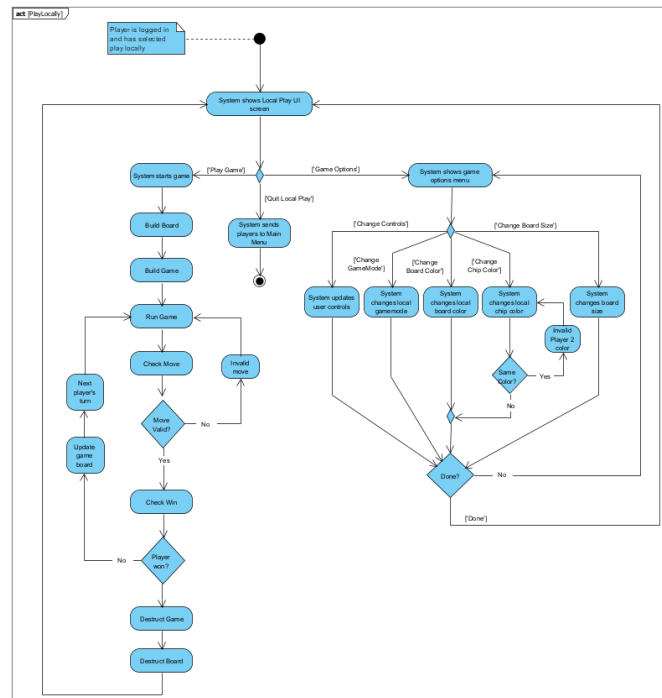
Final Class Diagram

Activity Diagrams

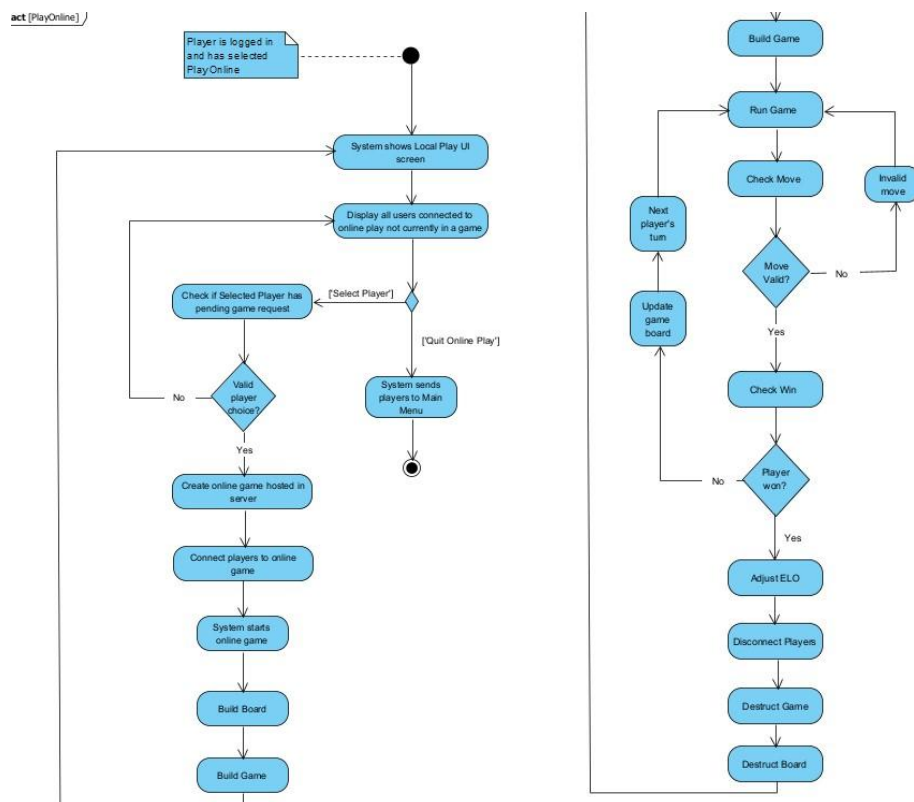
Provided below are activity diagrams created to demonstrate various expected situations in the project. Diagrams cover activities such as logging in, playing each gamemode, and a high level playing the game diagram to highlight the expected flow of the project.



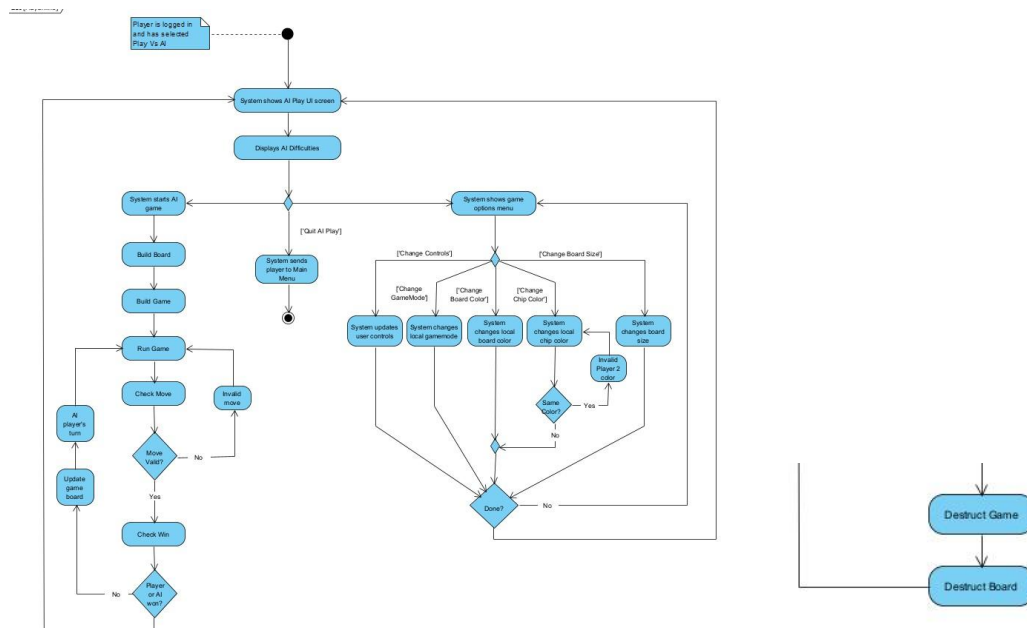
Logging In Activity Diagram



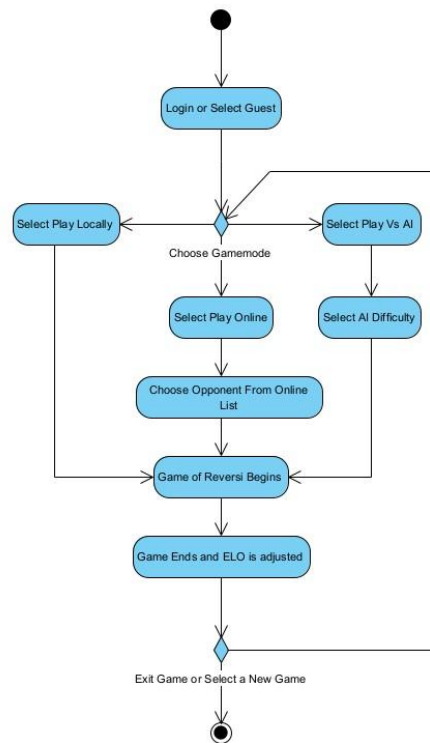
Play Locally Activity Diagram



Play Online Activity Diagram



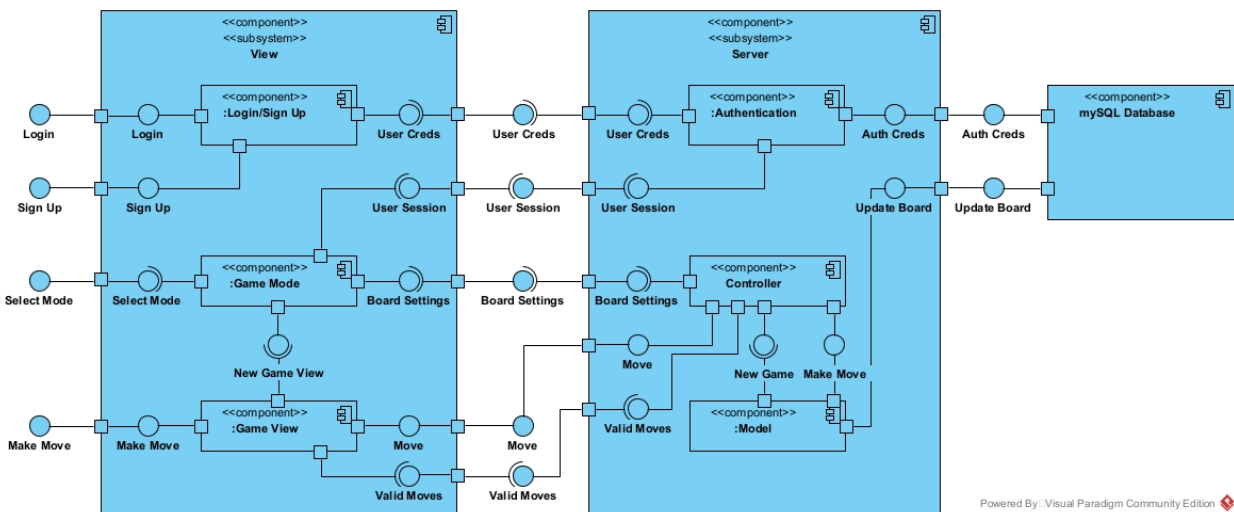
Play vs AI Activity Diagram



High Level Playing Game Activity Diagram

Component Diagrams

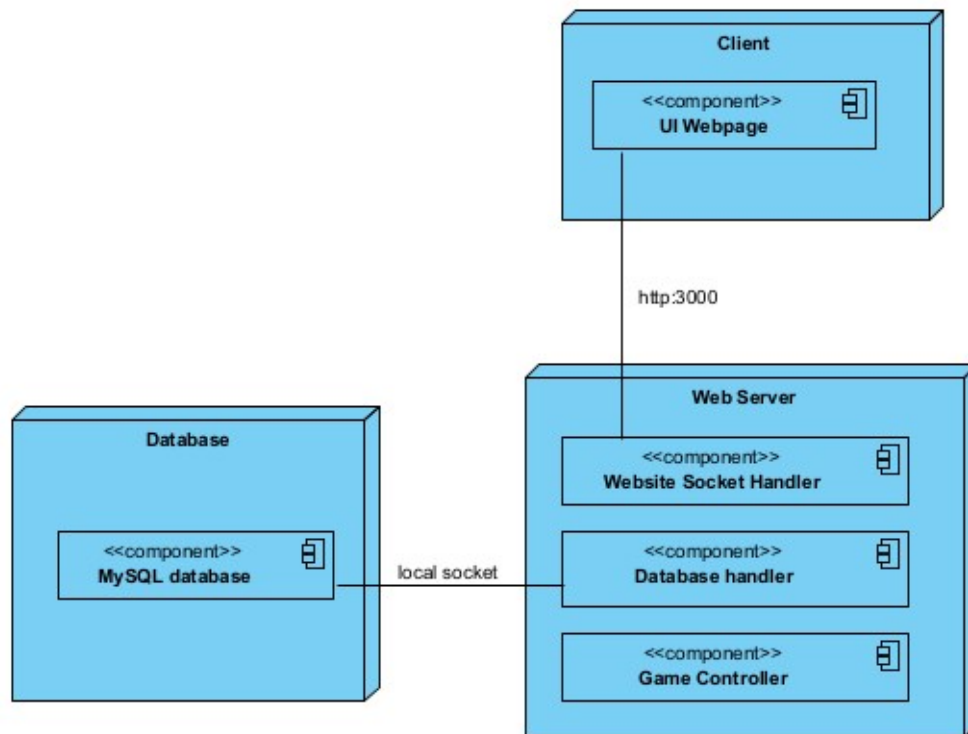
The component diagram was created to highlight the subsystem decomposition of the overall project. It highlights how primary abstract components in the system interact with each other to form the greater system. The component diagram is shown below.



System Component Diagram

Deployment Diagrams

The deployment diagram shows the interaction between our client, server, and database. The UI webpage connects to the website socket handler that manages user data or game moves sent through it. The server connects to the MySQL database through the database handler and also contains the game controller to manage game logic



Deployment Diagram

Design Patterns

Facade:

- Used while handling a player move to abstract out checking for a valid move
- Valid move checks if the space is open, if there are valid endpoints for the player's move, and if the space is in the valid found moves array
- The facade implementation is under the method in game that checks for each of the following returns
- If any of those checks returns false, then the entire function returns false on that move and only returns true otherwise

Prototype:

- Used to deep copy board for AI game implementation
- AI game needs to copy the board to test moves on it
- AI extends prototype so it can use its functions that allow it to deep copy an object

Chain of Responsibility:

- Used to pass off move down chain of responsibility from controller
- When a move is made client-side it gets sent to the controller
- The controller then sends the move down the chain to the game
- The game then sends the move through the game logic and to the board where it can then be applied to the board

Observer:

- Used to pass the board and the player object each turn to the database
- The database board class is a subscriber
- The player class is a subject
- The board class is a subject
- Every turn, the board and player notify the database that a change has been made and send board and player information to the database
- This reduces the number of dependencies and allows for further expansion of input objects to the database in the future

Test Plans

When planning the testing, we wanted to make sure that we covered all of the functions in our classes, but on a higher level as well. To do this, we also tested methods from the controller so that we could check that our high level methods worked as well. Aside from that, we had already done mock testing for some of the earlier model files. This mainly consisted of writing an example board and running the functions on it to make sure that the output was what we expected. So moving forward to write the actual tests, we will incorporate some of those preexisting examples.

Git Revision History

All Git revision history can be found on the Group 2 Reversi GitHub page here <https://github.com/dduuukk>.

Summary of Code Testing

- Test suites run through the Jest package
- Each model class has its own test suite
- Test suites test all of the functions in that class

- Tests check edge cases when applicable like checking if a move is valid or not
- Tests all pass and prove functionality of our system


Sprint Planning Meeting Notes

Sprint planning occurred at the beginning of each Sprint, and allowed the group to further define tasks and divide work. Shown below are all of the sprint planning meeting notes taken over the course of the semester.

2023-01-20 Meeting notes

 Date


20 Jan 2023

 Participants

- @Christian
- @Luke Wisner
- @Owen Zhang

 Goals


- Set objectives for Milestone 1
- Brainstorm diagrams
- Split diagram work

 Discussion topics

Time	Item	Presenter	Notes
	Milestone 1 Requirements		<ul style="list-style-type: none"> • Use case diagrams <ul style="list-style-type: none"> • Parent use case description • Other? • Ask Professor about how many specific use case diagrams we need <ul style="list-style-type: none"> • One for playing the game? • Logging in? • Activity/Communication Diagrams <ul style="list-style-type: none"> • Logging in • Play locally activity • Play online activity • Online matchmaking communication • Update user information activity • Class diagrams <ul style="list-style-type: none"> • Parent class diagram

 Action items

 @Christian Ask Professor about how many and what sub-use case diagrams we need

 Decisions

2023-02-03 Meeting notes

 Date


03 Feb 2023

 Participants

- @Christian
- @Luke Wisner
- @Owen Zhang

 Goals

- Find good tutorial resources for JavaScript and HTML
- Figure out best tasks for group members
- Split MVC

 Discussion topics

Time	Item	Presenter	Notes
	Milestone 2 Requirements		<ul style="list-style-type: none">• All logic happens in model• View only prints the changes that the model returns• Whole view and model go through controller• Model<ul style="list-style-type: none">• Base it off of Python example shown in class• Board class• Player class• Game logic class• Game class• View<ul style="list-style-type: none">• Webpage design and structure• Luke has experience with HTML and CSS• Board display• Pieces display• Display win screen• How do we get click from board to register move?• Controller<ul style="list-style-type: none">• Controls all game actions given from main

 Action items

- ☐ @Owen Zhang do JavaScript tutorials or HTML tutorials
- ☐ Look into Oden project
- ☐ @Christian do JavaScript tutorials
- ☐ Start the model
- ☐ @Luke Wisner start the view pages, functions, and CSS

 Decisions

 Luke focuses on the view for this milestone

2023-02-27 Meeting notes



Date

27 Feb 2023



Participants

- @ Christian
- @Luke Wisner
- @Owen Zhang



Goals

- Figure out which database to use
- Plan for AI
- ERD
- Make changes to show valid moves in GUI



Discussion topics

Time	Item	Presenter	Notes
	Milestone 3 Requirements		<ul style="list-style-type: none">• mySQL database<ul style="list-style-type: none">• npm install mysql• Easy integration• Examples from class• Chris can start on the database stuff with tutorials• AI<ul style="list-style-type: none">• Minmax algorithm• Check Professors slides• Luke is interested• Create the database and send the code to all users so that we can all have the same database



Action items

- ☐ @Owen Zhang look into SQL, create DB and send the DB code to the group
- ☐ @ Christian do tutorials for node's mysql and then integrate database into MVC
- ☐ @Luke Wisner start MinMax AI



Decisions



mySQL DB

2023-03-24 Meeting notes

 Date

24 Mar 2023

 Participants


- @Christian
- @Luke Wisner
- @Owen Zhang

 Goals


- Which design patterns will integrate the best into the current design


 Discussion topics

Time	Item	Presenter	Notes
	Milestone 4 Requirements		<ul style="list-style-type: none">• Possible design patterns<ul style="list-style-type: none">• Observer• Proxy• Facade• Chain of responsibility• Decorator• Adapter• Prototype• Re-do the whole setup to use client server<ul style="list-style-type: none">• Express js• Socketio


 Action items


- ☐ @Christian Bender learn express and look into socket.io

 Decisions

 Observer for board and player

 Prototype for AI deep copy

 Facade for making moves

 Chain of responsibility for selections and making moves

2023-04-10 Meeting notes



Date

10 Apr 2023



Participants

- @ Christian
- @Luke Wisner
- @Owen Zhang



Goals

- Figure out everything that is left in the project
- Online multiplayer



Discussion topics

Time	Item	Presenter	Notes
	Milestone 5 Requirements		<ul style="list-style-type: none">• Online multiplayer<ul style="list-style-type: none">• Pretty much all express stuff needs to be transferred to socket.io• Figure out the best way to implement• Figure out a way to have single player games be independent• Update saving to DB and pulling from DB for online• Update all View models back to HTML rather than EJS for the login and register pages



Action items



Decisions



Make new game objects in each socket connection to let single player gamemodes work properly

Product and Sprint Backlogs

Unfortunately, Jira sprint backlogs were not documented over the course of the semester and now access to each backlog has been limited by Jira. However, Jira does make accessible a history of changes to each of the Sprint backlogs. The tasks at the end of each sprint are shown below, and the link to the Jira where this information can also be found is

<https://cvbender.atlassian.net/jira/software/projects/RF23T2/boards/1>.

Wed, Feb 01 2023,
4:37pm

Sprint completed

RF23T2-3 Parent Use Case Diagram
RF23T2-4 Parent Use Case Description (Text Steps)
RF23T2-5 Logging In Activity Diagram
RF23T2-7 Online Matchmaking Communication Diagram
RF23T2-8 Update User Information Activity Diagram
RF23T2-10 Define Scenarios at Office Hours
RF23T2-11 Class Diagram
RF23T2-12 Play Locally USE CASE
RF23T2-13 Play vs AI USE CASE
RF23T2-14 Start game USE CASE
RF23T2-15 Play online USE CASE
RF23T2-17 Playing the Game update diagram
RF23T2-20 Play Locally Activity Diagram
RF23T2-21 Play Online Activity Diagram

Milestone 1 Sprint Backlog (Finished Tasks)

Mon, Mar 13 2023,
9:37am

Sprint completed

RF23T2-23 Board Class (Model)
RF23T2-28 Player Class (Model)
RF23T2-32 Game Logic Class (Model)
RF23T2-37 Update Class Diagram
RF23T2-38 Game Class (Model)
RF23T2-42 Webpage design and structure (View)
RF23T2-43 Board Display (View)
RF23T2-44 Display Pieces (View)
RF23T2-45 Display Scores (View)
RF23T2-46 Display Win Screen (View)
RF23T2-47 Main file to run in webpage
RF23T2-48 GameController Functions (Controller)
RF23T2-49 Get Click from board to register move (View)

Milestone 2 Sprint Backlog (Finished Tasks)

Mon, Apr 03 2023,
5:03pm

Sprint completed

RF23T2-56 Re-write code to remove all statics from each class
RF23T2-57 Re-write flip pieces to use vectors
RF23T2-58 Re-write find endpoints to use vectors
RF23T2-59 Re-write check moves functions to display pieces and use vectors
RF23T2-60 AI Class
RF23T2-62 ERD
RF23T2-63 MySQL sql code
RF23T2-64 Connect AI to Controller
RF23T2-65 AI difficulty selection page
RF23T2-66 Register New User Page
RF23T2-67 Update Main to prep for handling database function in our html webpages
RF23T2-68 DB Interface Class
RF23T2-69 DB Board Update / Read Classes
RF23T2-70 DB Player Login / New User Classes
RF23T2-71 Connect DB to model
RF23T2-72 Update Class Diagram 3/20/23

Milestone 3 Sprint Backlog (Finished Tasks)

Sun, Apr 16 2023,
3:36pm

Sprint completed

RF23T2-73 Prototype Design Pattern
RF23T2-74 Observer Design Pattern
RF23T2-75 Facade Design Pattern
RF23T2-76 Handler Design Pattern
RF23T2-77 Setup express server
RF23T2-78 Link server to client for onclicks
RF23T2-79 Link database functionality
RF23T2-80 Link AI functions to server
RF23T2-81 Get database readings for creating new accounts
RF23T2-82 Class Diagram 4/6

Milestone 4 Sprint Backlog (Finished Tasks)

Mon, Apr 24 2023,
8:45am

Sprint completed

RF23T2-83 Fix router to server connection
RF23T2-84 Create online connection between players
RF23T2-85 Create individual sockets for players who sign in
RF23T2-86 Match players if they've selected the same settings
RF23T2-87 Create online game pages and messages
RF23T2-88 Create testing suite for model
RF23T2-89 Update database with online board
RF23T2-90 Rejoin exited game from modal

Milestone 5 Sprint Backlog (Finished Tasks)

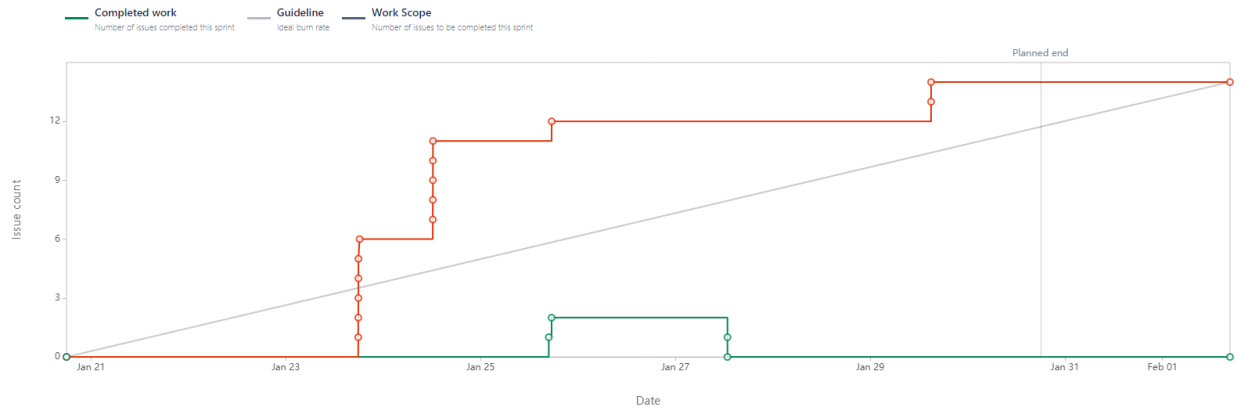
The final project backlog is shown below.

▼ Backlog (5 issues)		0 0 0 Create sprint
RF23T2-63	MySQL sql code	IN REVIEW 0%
RF23T2-91	Leaderboard	TO DO
RF23T2-92	Change colors	TO DO
RF23T2-93	Pull from DB on other game modes than local multiplayer	TO DO
RF23T2-94	Make online use only one game object	TO DO

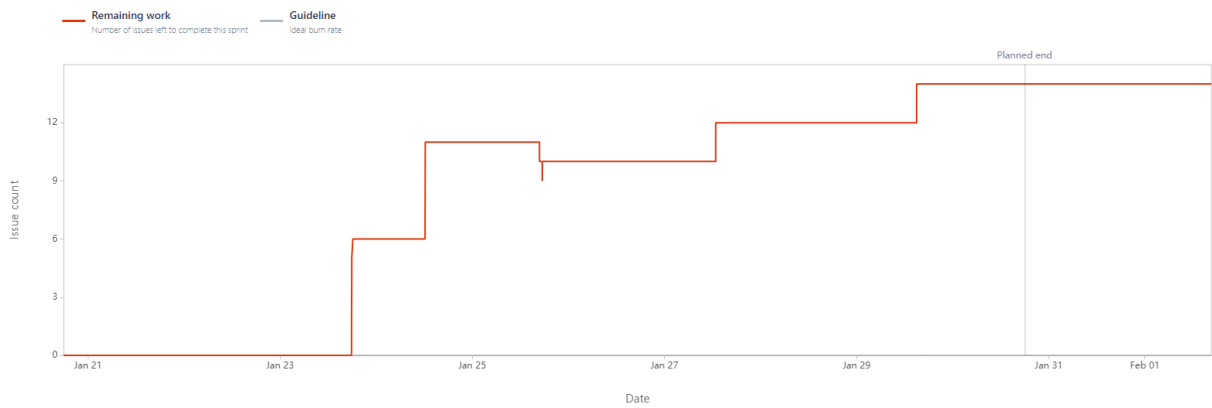
Final Project Backlog

Burnup and Burndown Charts

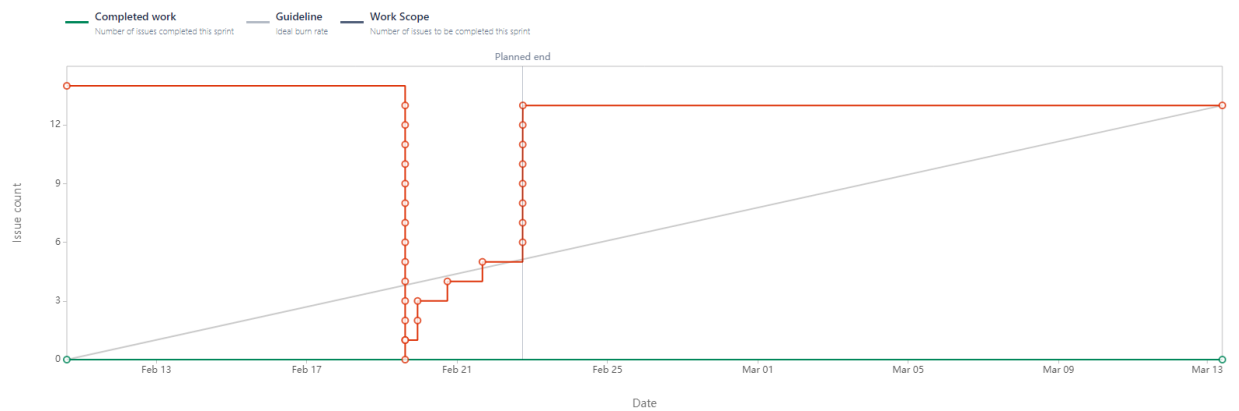
Throughout the course of this semester, Jira generated burnup and burndown charts for each milestone of the project. However, for some reason, Jira did not mark tasks as completed, even when the group made sure that all tasks that were completed were dragged into the completed section of the board tab. Importantly, this means that all burn up and burn down charts do not account for the team's progress throughout each milestone sprint. However, the team tried to follow the estimates given in the charts, and while the charts do not show completed work, the tasks were completed. This still means that the burndown charts do not show any of the tasks being completed. The burnup and burndown charts are shown below.



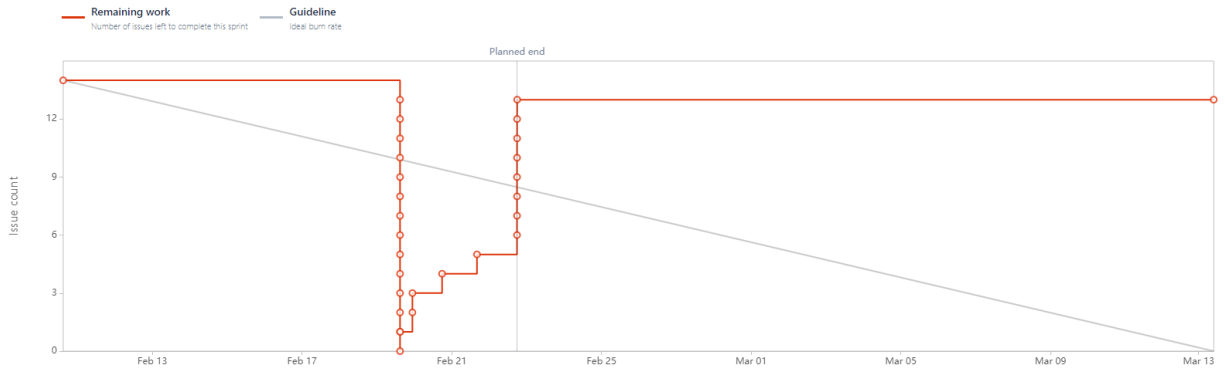
Milestone 1 Burnup Chart



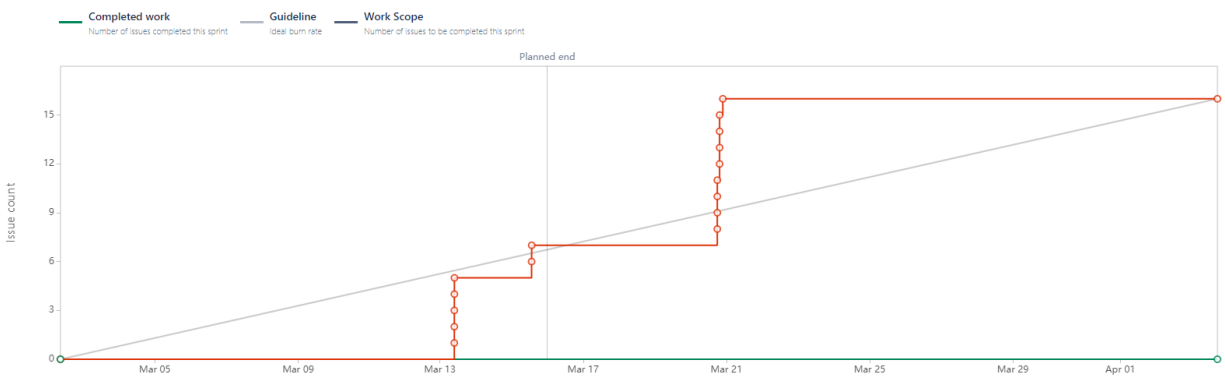
Milestone 1 Burndown Chart



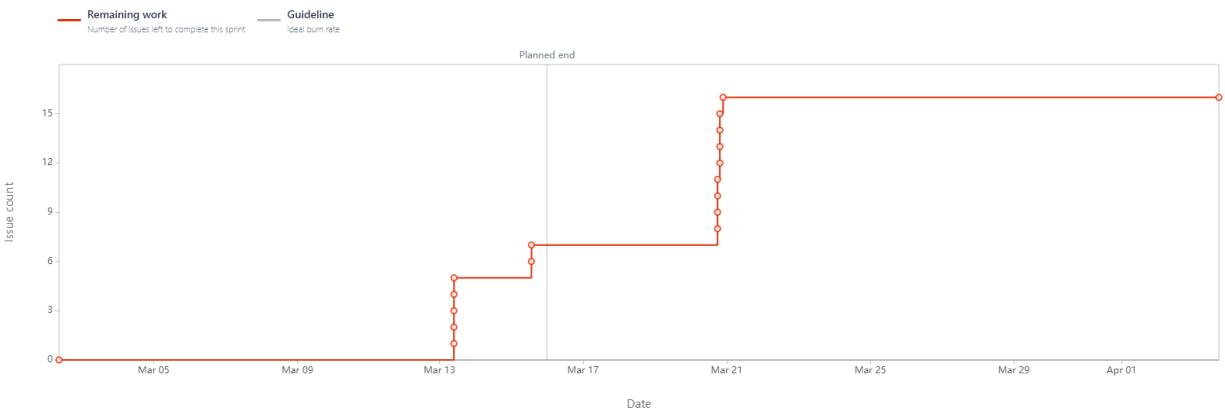
Milestone 2 Burnup Chart



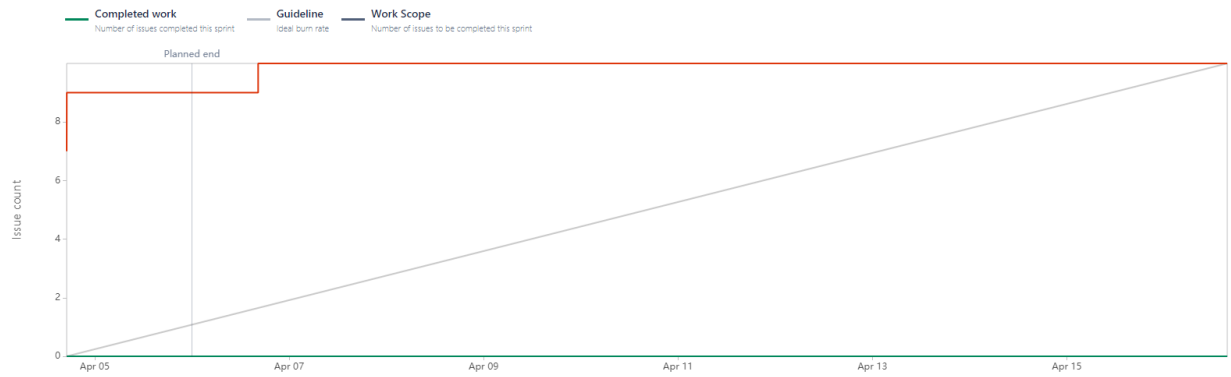
Milestone 2 Burndown Chart



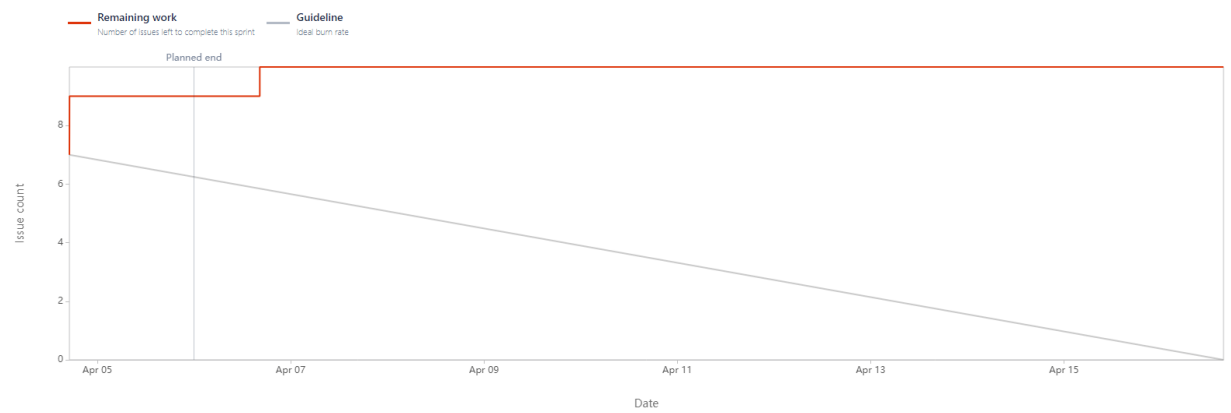
Milestone 3 Burnup Chart



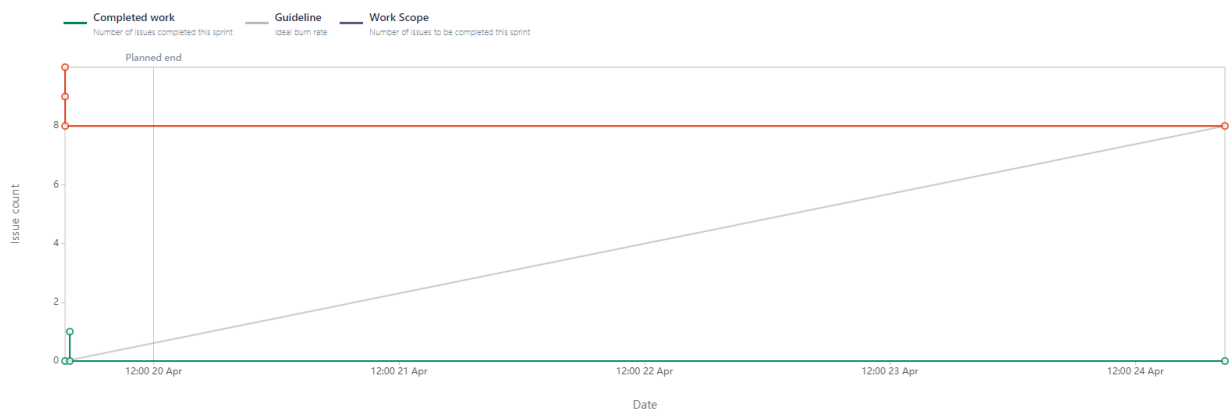
Milestone 3 Burndown Chart



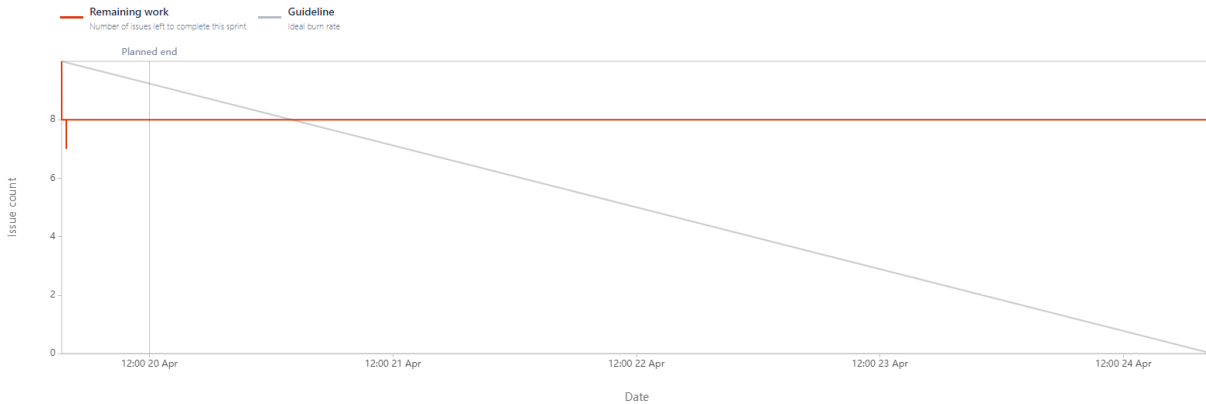
Milestone 4 Burnup Chart



Milestone 4 Burndown Chart



Milestone 5 Burnup Chart



Milestone 5 Burndown Chart

Project Timeline

The project was completed in a total of 4 months, with the following major milestones and deliverables:

- Project planning and requirements analysis: 1 month
- Design and development of core features: 2.5 months
- Testing and refinement: 1 week
- Deployment and final testing: 1 week
- Sprint length: 2 weeks

Accomplishments

The project achieved the following major accomplishments:

- Successful development and deployment of a fully functional online Reversi program
- Implementation of a reliable user authentication and registration system connected to a database
- Implementation of various design patterns for greater flexibility
- Development of an AI-powered single-player gameplay mode with varying difficulty levels
- Flexible, modular code base

Challenges

The project encountered several challenges during its development and deployment, including, but not limited to:

- Using javascript, node.js, MySQL, Socket.IO, all of which was largely unfamiliar
- Implementing the MVC architecture
- Connecting the client-server-database
- Creating the AI with more complex heuristics for difficulty

These challenges were addressed through continuous testing and refinement, and collaboration with the project team.

Future Work

While a lot was accomplished throughout this project, there were still features and ideas that were not implemented in the system due to timing constraints. These features are listed below.

- Break the server into reusable functions
 - As of now, the server is less flexible and readable
 - Different game modes have reusable steps
 - These can be abstracted into separate functions
- Add game recall to online multiplayer
- Add dynamic room browser
 - Online play is hosted in socket rooms, so it has the capacity to host many parallel online games
- Add a more in depth ELO system
 - Current ELO writing is proof of concept (will use npm elo on suggestion of Professor)
- ELO based matchmaking
- Change the online multiplayer matching to have only one game object shared between two computers, rather than two game objects and two controllers
 - This would ensure synchronization across online multiplayer games

Overall, there are still a number of features that the team would like to implement to bring full functionality to the game. The systems are in place to provide these features, making updates easier moving forward.

Lessons Learned

This project was very challenging at points, but it was even more rewarding for how much we learned and how much more confident we became in our coding abilities. Coming into this project, most of us didn't have web application experience or even knowledge of JavaScript.

Luke was the only one who knew any html/JavaScript, so the rest of the group had to learn throughout the semester. Going from no JavaScript knowledge to building a full client-server-database connection with node packages is impressive and demonstrates how important being able to quickly learn new coding skills is.

Due to our lack of web application knowledge, we all had to learn a lot of new packages and libraries to make our code run. None of us had knowledge of socket.io, express, database setup, database communication, or even using modules in javascript, so there was a lot to learn. By pushing us out of our comfort zone and forcing us to quickly teach ourselves new coding practices, this project showed that we're capable of working as software engineers. Along with this, we also learned many helpful code structures and design patterns that we can implement in future projects.

Finally, the group learned a lot about good software engineering practices. There were many points throughout this semester where we looked back and thought, "if only we had designed the system in this way, this future issue would have been easier to work with". These experiences enforced that software engineering practices which revolve around planning and making the system flexible for future changes are super valuable, teaching us that taking the correct time to plan and document, as well as taking the time to ensure the system is flexible and designed with good practices is more important than just having knowledge of a coding language.

Conclusion

In conclusion, the project to develop an online Reversi program was successful in achieving many of its goals and objectives. The program provides a user-friendly and interactive platform for playing the game online, with a comprehensive set of features and functionality. The program was designed to handle multiple simultaneous players and provide a seamless gameplay experience. It was built using various techniques learned in class, implementing modern system, object, and database designs. We used UML to plan out the project, and built it using the MVC architecture. Throughout the development phase, we implemented flexible, modular code with various design patterns to create a base which can be built upon in the future for any features we want to add later on. We recommend further improvements in performance and scalability, and regular updates to ensure that the program remains competitive and engaging for users. Overall, we are proud of our accomplishments and look forward to continued success with future software development projects.