

Benjamin Hoang

Daniel Burris

Irene Kasian

Zachary Haynes

Team 8 - CS 321

Ultimate Tic-Tac-Toe

# Contents

[Contents 1](#_Toc467427970)

[Project Description 2](#_Toc467427971)

[Project Management 3](#_Toc467427972)

[Use Cases 4](#_Toc467427973)

[Requirements 5](#_Toc467427974)

[Use Cases 7](#_Toc467427975)

[Implementation 7](#_Toc467427976)

[Discussion 8](#_Toc467427977)

# Project Description

Our project is to create ultimate Tic-Tac-Toe. Some of the functionality for this game will be: player vs. player, simple computer AI, database management for features such as: highscore, leader board, and user profiles, menu with options to play the game, change settings (such as color schemes), load a user profile, and close the application.

# Project Management

## History

Throughout the semester, we have worked to develop an application that will allow a user to play Ultimate Tic Tac Toe against a friend or against a very simple AI. We first began with creating a GUI prototype of a menu and a game board. After creating the prototypes, we set to implement the main controls of the game, such as enabling/disabling pressed tiles and available boards, as well as checks for a game winner. During this stage, we also added a few settings options. Next, we worked on calculating highscores. While working on that, we realized that it may be useful to create user profiles that will all for easier storage of settings, and the amount of games played, etc. The links below lead to our Github repository for more details.

[Github Commits](https://github.com/BenjHoang/Ultimate_tictac) [Github Repository](https://github.com/Dburris13/UltimateTicTacToe)

## Personnel

**Daniel Burris** - Senior in Computer Engineering, 3 years experience with C++ / OOP.

**Ben Hoang** - Senior in Computer Science, 4 years practice C++/Java, Algorithms.

**Irene Kasian** - Junior in Computer Science, 2 years experience with C++

**Zach Haynes** - Junior in Computer Engineering, 4 years experience with Java/OOP.

## Effort

On average, approximate 6 hours per week were dedicated to this project between all 4 of us.



# Use Cases

## Main use scenario

## Load player profile scenario

# Requirements

## Overview

This game has to address problems such as file based database management, dealing with GUI design and simple animations, and working with java. In addition, this game needs to appropriately disable taken tiles, and enable/disable the correct boards a player can play on. The game also needs to be able to implement the rules of basic Tic Tac Toe, and Ultimate Tic Tac Toe.

It is assumed that the user knows how to play the game. The program will allow a user to play as many game of ultimate tic tac toe as they desire against another player or against a very simple AI.

## Defined requirements

**GUI** - Our application is GUI-based, allowing the user to play the game, change settings, load profiles, and check the high scores all from a GUI.

**Text formatting and processing** - We use HTML to format a few JLabels; we also use group layout to organize text.

**Graphics** – We have a main menu that features a simple background animation, and our game scene is created and manipulated through our Game hierarchy classes.

**Storage and retrieval of information** - we have a file based user profile system that creates a file for each user as well as a high score system that is also stored as a file.

**Editing and configuring the software product** - We have a settings menu that allows the user to customize the color scheme and resolution of the game board.

## Project specific requirements

1. have a basic AI to play against the user
2. The user is able to play the game with a friend or against the AI
   1. This including having correct enabling and disabling of the mini tic tac toe boards and not allowing a user to overwrite a previously played on tile.

## Future modification and extensions

1. AI difficulty settings
2. Saving/Loading an unfinished game
3. More customization options, such as color schemes, and fonts.
4. Randomly selecting which player goes first/which character they are (X or O)

## Indexed summary list

Software with provide a basic user interface for menu navigation.

Software will feature a fully functioning Ultimate Tic-Tac-Toe game.

Software will be configurable based on user settings.

User settings as well as game statistics can be saved / loaded into user profiles.

Software will feature a high score system.

Software will feature basic animation.

## Associated tests

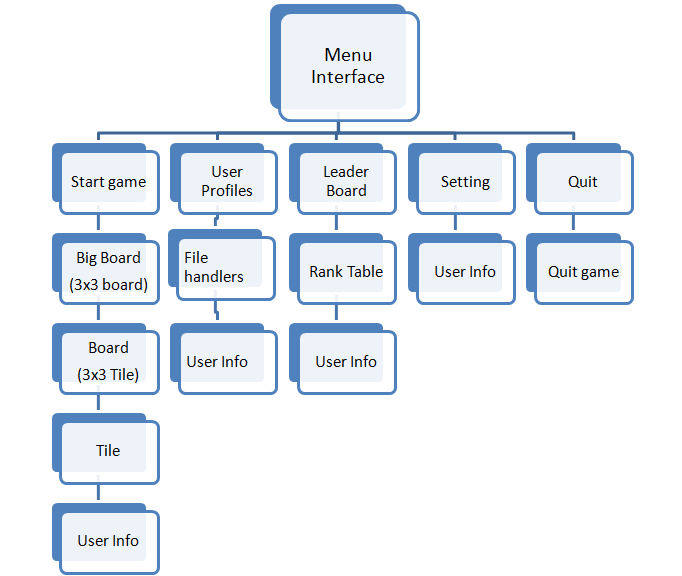
|  |  |
| --- | --- |
| GUI | Enter into all menu branches and return to main menu. |
| Text Processing | Visually verify that all menus look correct. |
| Graphics | Visually verify that game scene looks correct and main menu animation is correct. |
| Storage and Retrieval of Information | Load user profiles, change user profiles, load high score table, complete game, load high score table again. |
| Editing and Configuring the Software Product | Change settings with user profiles and without user profiles and enter game scene. |
| Artificial Intelligence | Play several games against the AI. |

# Design

## Overview

\*\*Basic deciption of how the model works. Include CRC cards?

## Data Model: Object and State Design



## Views and Controls

??????

# Implementation

## Packages and classes

1. **Resources package** – this package contains the gifs used for the dialogs; this would be a good place to put images that can be used to further customize game appearance
2. **MenuScenes Package** – this package contains the Jframes used in the application (except for the game board)

The following classes are included in the package:

* 1. MenuGUI
  2. Rank\_Table
  3. UserProfiles
  4. Settings
  5. Dialog (Abstract)
     1. WinDialog
     2. TieDialog

The MenuGUI is the main JFrame. From the MenuGUI, the RankTable, UserProfiles, or Setting JFrame can be called.

The Dialog is an abstract class for popup dialogs when a game ends. It has two implemenations, WinDialog, and TieDialog. Whenever a player or AI wins a game, WinDialog class is called. Whenever the game ends in a Tie the TieDialog class is called.

1. **GameFiles Package** – this package contains all the classes that define the game GUI and the game logic

The following classes are included in the package:

* 1. BigBoard
  2. Board
  3. Tile
  4. Game

When a game starts, the class Game is created. Within the Game class, there is a BigBoard Class that defines the entire Ultimate Tic Tac Toe board. The BigBoard class encapsulates 9 Board classes, which represent a regular Tic Tac Toe board. Each board class encapsulates 9 Tile classes, which represent a single space on a board.

1. **PlayerInfo Package** – this package contains all classes related to the player

The following classes are included in the package:

* 1. UserInfo
  2. Player
  3. FileProfile
  4. FileManager(?)

The UserInfo class represents all data associated with a user’s profile. The FileProfile class is an implementation of the FileManager class. The purpose of FileProfile is to write and read player’s profiles. The Player class is used to keep track of the current player’s turn.

1. **GUIMisc Package** – this package contains all miscellaneous classes that can be used with a GUI

The following classes are included in the package:

* 1. AnimatedPanel
  2. BackgroundChars
     1. XChar
     2. YChar

The Animated Panel class defines a JPanel that holds implementations of the BackgroundChars class. The BackgroundChars class is an abstract class that defines a character to be displayed.

1. **AI Package** – this package contains all classes used to control the AI player.

The following classes are included in the package:

* 1. AI

The class AI defines how an AI will choose where to make its move.

For more information about each class see the Classes section below, or Javadocs

## Application program interfaces

## Classes

1. AI -- AI (Computer Player) Class. This is created if a game is started with the AI radio button checked.
   1. Contains a function that chooses which tile the AI will select
2. AnimatedPanel -- A JPanel that contains the animated X's and O's displayed on the GUIMenu.
   1. Creates and array list of 6 useable colors
   2. Creates an array list of BackgroundChars that will be displayed
3. BackgroundChars -- An abstract class defining the characters that will be displayed in the AnimatedPanel class
   1. Picks a random starting location for the characters to be displayed
   2. Picks a random amount of time to wait before drawing the character
4. XChar -- A class that creates an animated X character.
   1. Sets which string/character is displayed (in this case an X).
   2. Defines vertical movement for the X.
5. YChar -- A class that creates an animated Y character.
   1. Sets which string character is displayed (in this case Y).
   2. Defines horizontal movement for the Y
6. BigBoard -- This class is the BigBoard class that encapsulates 9 Board classes.
   1. Checks for win cases
   2. Visually identifies won or tied boards
7. Board -- This class represents a Board of 9 Tiles.
   1. Keeps track of the return status of its tiles
   2. Checks if the board is won
8. Game -- This class represents our game scene.
   1. Implements BigBoard Class
   2. Updates user profiles
9. Tile -- class that extends the JButton. Represents a single space in a tic tac toe board
10. Dialog -- This is the abstract class of the dialog box that pops up after the game as ended.
    1. Defines all components that all dialogs should have, such as a return to menu button
11. MenuGUI -- This class represents our main menu scene. Allows the user to navigate to all parts of the application
12. Rank\_table -- This class represents our high scores scene.
    1. Will rand the User Profiles appropriately
13. Settings -- This class represents our user settings scene.
    1. Allows the user to change color schemes and resolution
14. TieDialog -- This is an implementation of the Dialog class to display when a game is tied.
15. UserProfiles -- This class represents our user profiles scene.
16. WinDialog – This is an implementation of the Dialog class to display when a game is won by anyone, be it the Computer, or the player.
17. FileManager -- Abstract class defining abstract methods for all File I/O.
18. FileProfile – An implementation of FileManager. Defines file I/O for userProfiles.
19. Player – a class for keeping whose turn it is.

UserInfo -- This class tracks on User Information, such as the user’s preferred settings options, and the amount of games they’ve played.

For detailed information about classes’ functions and variables, see javadocs.

## Test plan

1. Ensure all GUI’s render correctly, will all components in the correct places.
   1. Tested by creating a list of all GUI’s and which components each should have, and sketching an approximation of what they should look like. Visually made sure the list and sketch matched what was rendered
2. Ensure each GUI’s buttons perform appropriate actions.
   1. Tested by manually going through each clickable option in the GUI.
   2. Had to make sure that when moving to a new JFrame, the previous one was disposed.
   3. Had to make sure the new JFrame that was loaded is the one what we selected.
3. Ensure the tiles of the game board disable when click, and display the correct mark
   1. Tested by clicking on a tile, and visually ensuring the appropriate mark appeared.
   2. Continued to test by clicking more tiles and ensuring the mark changes every turn.
4. Ensure that a board will disable when it is won, or tied. Also, make sure that the board is colored according to its status
   1. Tested the model’s logic by playing the small tic tac toe boards making sure each time there were 3 x’s or 3 o’s in a row in any orientation, the board disabled and painted itself according to the winner
   2. Purposely tied the tic tac toe board to ensure the board still disables and is painted all red to indicate tie.
5. Ensure game recognizes when there are 3 boards won by the same player in a row
   1. Tested by trial and error like previously described, but instead of ensure the game board is painted, we ensure the appropriate JDialog appears
6. Ensure that new profiles are created correctly and can be loaded
   1. Created many dummy profiles to and changed settings in each to ensure all information is written to the file.
   2. Loaded dummy profiles, and played games to ensure the profiles loaded correctly and new information was saved.
   3. Reopened application, loaded dummy profiles, and visually made sure the previously selected setting display.
7. Opened leaderboard to ensure it was keeping track user’s win information.

## Tested functionality

1. Menu GUI renders correctly.
2. Selecting Start Game will create a new game, and initialize all components correctly.
3. Game board’s tiles will disable when pressed. Each tile will remain disabled for the duration of the game.
4. Each board will disable for the duration of the game when it is won or tied and color the tiles according to the winner of the board.
5. If there are 3 Boards won by a single player in a row, a modal congratulations dialog will appear.
6. If there are no 3 Boards in a row for a winner, a modal dialog for a tie will appear.
7. User can return to menu in the middle of a game.
8. User can exit application in the middle of a game.
9. User Profiles Button will load the UserProfiles GUI.
10. When User Profiles is selected, a directory will be created to store the user profile files if it does not already exist.
11. In the UserProfiles GUI, the Load User button loads the selected profile.
12. In the UserProfiles GUI, the New User button will prompt the user for a name
    1. That name will be added to the list
13. Return button in the UserProfiles GUI returns the user to the Menu
14. When the Leaderboard button is pressed, it will load the Rank\_Data GUI.
15. The Rank\_Data GUI displays all users, and their scores.
16. The leaderboard data will be stored in a file
17. Leaderboard data will be saved and available the next time the application is started.
18. Settings button will load the Settings GUI.
19. Settings options will save correctly, provided there is a valid user profile loaded, and be available the next time the application is started.
20. Quit button exits the application.

## Untested functionality

No untested functionality.

## Requirement satisfaction

# Discussion

As with most of the projects in this class, the majority of our trade-offs revolve around schedules and time consumption. Would we rather spend time writing a complicated and creative computer opponent, a project specific requirement, or spend the same amount of time creating an animated login screen, a project requirement. Our code is certainly not the most abstract / generic it can be, but we tried our best to create general methods and organize private and public variables so that a potential future team would be able to look at our code and figure it out. There is some complicated logic in some areas of our code that could be simplified / rewritten, but we tried to balance out the complication with detailed javadocs.