

Team 8 - CS 321

11.29.2016

Benjamin Hoang

Daniel Burris

Irene Kasian

Zachary Haynes

Ultimate Tic-Tac-Toe

Copyright 2016 Benjamin Hoang, Daniel Burris, Irene Kasian, Zachary Haynes. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3, (3 November 2008) or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license can be found at <http://www.gnu.org/copyleft/fdl.html>.

Contents

[Project Description 2](#_Toc468130030)

[Project Management 3](#_Toc468130031)

[Use Cases 4](#_Toc468130032)

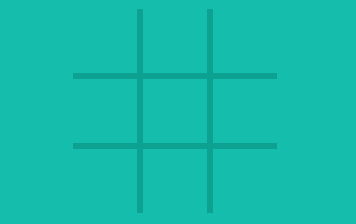
[Requirements 5](#_Toc468130033)

[Design 6](#_Toc468130034)

[Implementation 7](#_Toc468130035)

[Discussion 12](#_Toc468130036)

# 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 high scores. 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. You can see our progress by looking at the commits we made on the GitHub repos [here](https://github.com/BenjHoang/Ultimate_tictac) and [here](https://github.com/Dburris13/UltimateTicTacToe). The reason we have two GitHub repositories is because as well as learning java this semester we learned GIT, managing all 5 branches on the first repository was becoming a little too much to handle so we swapped to a clean state.

## Personnel

**Daniel Burris** - Senior in Computer Engineering, 3 years’ experience with C++ / OOP. Worked on PlayerInfo, MenuScenes, GameFiles, AI.

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

**Irene Kasian** - Junior in Computer Science, 2 years experience with C++. Worked on GUIMisc, resources, GameFiles.

**Zach Haynes** - Junior in Computer Engineering, 4 years experience with Java/OOP. Worked on AI, general code cleanliness.

## Effort

Each “release” for us would be a **commit**. In the first repo, each commit was approximately a weekends, worth of time. In the second repo, we tried to commit any changes me made, so depending on the complexity of the change, it could range from a documentation update (10 minutes) to a major bug fix (3+ hours).

The final submission (Documentation, Javadoc, Bug checking / final test) took about 10 hours between all four of us.

On average we spent ~ 8 hours each week between all 4 of us + 2 hours for bi-weekly meetings, so total project time would be (8\*10 + 2\*5 + final submission time (10)) = 100 hours.

# Use Cases

## Use Case 1 : Main Use Scenario

## Use Case 2 : Load Player Profile Scenario

# Requirements

## Overview

This game has to address problems such as file centric database management, good GUI design, clear and appropriate animation, game mechanics / quality, ranking system, basic AI. The program provides an entertainment platform for the end user, nothing spectacular which results in hundreds of games hours, but serving as a simple distraction similar to Windows OS games (solitaire, hearts). The customers of the program is really anyone who needs to kill a couple minutes, the heart of our game isn’t something that is going to entertain someone for hours, but may give them a nice break from say, working on their senior design project. The game will run on any system with the appropriate java environment installed on it.

It is assumed that the user knows how to play the game and the environment they are working on has the correct environment installed. The program aids the end users through its graphical user interface, particularly in the game scene, it will highlight which board and tiles are playable and what the last move played was.

## Defined requirements

**GUI** - The application is GUI-based, allowing the user to play the game, change settings, load profiles, and check the high scores all from a GUI. All readily available on application start.

**Text formatting and processing** – The application use HTML to format a few JLabels; we also use swing GUI group layouts to organize text.

**Animation and Graphics** – The application have a main menu that features a simple background animation, and a game scene that is created dynamically based on user settings that provides clearly identifiably and recognizable graphics.

**Storage and retrieval of information** - The application has a file based user profile system that creates a file for each user and stores things such as user settings and information. It also features a file based high score system that creates its own data file for retrieval and storage of information and organizes its high scores based on how many moves it took to defeat his opponent.

**Editing and configuring the software product** – The application has a settings menu that allows the user to customize the color scheme and resolution of the game board.

## Project specific requirements

1. Giving the user the option to play versus a friend (locally) or versus an AI, albeit a stupid one. (All Use Cases)
2. Making a visually appealing menu screen with animations. (All Use Cases)
3. Creating a game board with easily recognizable graphics that visually present the user with the set of playable tiles and marks the last tile played with interchangeable colors and resolutions. (All Use Cases)
4. Creation and saving of User profiles that allows for the storing and retrieval of user settings and information. (Use Case 2)

## Future modification and extensions

1. AI difficulty settings (easy, normal, hard), our current AI is random.
2. Saving/Loading an unfinished game.
3. Additional ranking options.
4. More customization options, better visuals in general.
5. Dynamically assign each player to X or O, currently Player 1 is always X and always goes first.

## Indexed summary list

1. On application start, main menu appears. **(1.1)**
2. From the main menu the user is able to:
   1. Select whether he will player against another player or AI **(1.3)**
   2. Start the game with default settings and guest profiles **(1.4)**
   3. Load or Create a new user profiles which will track game statistics such as games played / won, and preferred settings. **(1.2, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7)**
   4. View current local high scores if any are present. **(1.11)**
   5. Change game settings and save game setting is user profile is loaded. **(Always Applicable)**
3. Once in the game screen:
   1. Player 1 (X) goes first **(1.6, 1.7)**
   2. Player 2 or AI then picks their move **(1.8, 1.9)**
   3. Repeat until win conditions are met **(1.10)**
   4. Appropriate message is drawn to screen **(1.12)**
4. Once out of the game scene depending on whether user profiles were loaded, results are saved in profiles and rank data and the user is returned to the main menu. **(1.12,1.13)**

## Associated Tests

|  |  |
| --- | --- |
| Defined Requirements |  |
| GUI | Application starts in main menu and all branches (settings, user profiles, etc.) load without fail. |
| Text Processing | Visual verification that text is aligned and good looking. |
| Animation and Graphics | Visual verification on opening animation and updating game graphics. |
| 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. |
| Specific Requirements |  |
| Swapping between player mode and AI mode | Load game with either option checked, verify correct results. Complete games versus the AI. |
| Visually appealing menu screen | Ask ourselves *Is it not visually appealing?* With a Maximus inflection |
| Game board with updating graphics | Play through a full game and verify that board is updating graphically correctly every time. |
| User Profiles | Create, load and edit several default user profiles. |

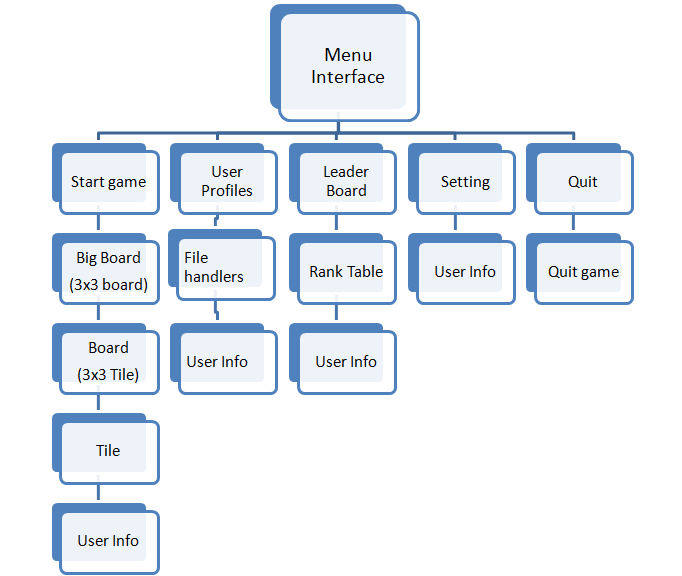
# Design

## Overview

A visual description of our system can be seen below in the Data Model section. Essentially, two UserInfo objects are passed around the entire system (from Main Menu to each class in MenuScene package) and determine visually what changes and how files (User Profiles / Rank data) are to be edited. If no user profiles are loaded manually, then the system uses the default UserInfo parameters to draw itself and perform file operations.

## Data Model: Object and State Design

We have a hierarchical data model as seen below. Each MenuScene item is passed and returns information to our Main menu which passes that information to the next MenuScene item. In each MenuScene it passes the information downs its hierarchy to whoever needs it which returns it back to the JFrame.



## Views and Controls

The views of our system are all Jframes. Each JFrame is accessible from the main menu and provides buttons with action listeners that allow the users to control our system. As previously stated all the controls of our system are through JButtons, each JButton has an associated action listener which allows the user to navigate the views, play the game, load/save profiles, load/save settings. Below you can see a list of all our CRC cards. Each class (card) is connected as a view (JFrame) or control (button) to our system.

|  |  |
| --- | --- |
| Tile (Control) | |
| Responsibilities: | **Collaborators:** |
| Advance game state/ change tile within board | Board |
| Set X or O depends on who clicked |  |

|  |  |
| --- | --- |
| Board (Control) | |
| Responsibilities: | **Collaborators:** |
| Returns its status to bigboard | Tile |
| Check if it is won by someone | BigBoard |
| Encapsulates 9 tiles |  |

|  |  |
| --- | --- |
| BigBoard (Control) | |
| Responsibilities: | **Collaborators:** |
| Encapsulate 9 boards | Board |
| Check for winner | Game |
| Paint board if it is won | Dialog – WinDialog, TieDialog |

|  |  |
| --- | --- |
| Game (View) | |
| Responsibilities: | **Collaborators:** |
| Define a GUI for the game | AI |
| Write player information | MenuGUI |
| Tell the AI to play if needed | UserInfo |
|  | FileProfile |
|  | Player |

|  |  |
| --- | --- |
| MenuGUI (View) | |
| Responsibilities: | **Collaborators:** |
| Acts like a controller for the user to get to all p arts of the application | Rank\_table |
| Begins background animation | Settings |
|  | UserInfo |
|  | Game |
|  | AnimatedPanel |
|  | UserProfiles |

|  |  |
| --- | --- |
| Settings (View) | |
| Responsibilities: | **Collaborators:** |
| Update user preferences | FileProfile |
| Tell the menu the user preferences | UserInfo |
|  | MenuGUI |

|  |  |
| --- | --- |
| AI (Control) | |
| Responsibilities: | **Collaborators:** |
| Select where to move | Board |
|  | Game |
|  | Tile |

|  |  |
| --- | --- |
| UserProfiles (Control) | |
| Responsibilities: | **Collaborators:** |
| Write new user information to file | FileProfile |
| Load user information to use | UserInfo |
| Return the user to menu |  |

|  |  |
| --- | --- |
| FileProfile (Control) | |
| Responsibilities: | **Collaborators** |
| Write to a file | FileManager |
| Read information from a file | UserInfo |

|  |  |
| --- | --- |
| FileManager – Abstract – (Control) | |
| Responsibilities: | **Collaborators:** |
| Read file | FileProfile -- implementation |
| Write file |  |

|  |  |
| --- | --- |
| RankTable (View) | |
| Responsibilities: | **Collaborators:** |
| Sort user high scores | UserInfo |
| Display the high scores | MenuGUI |

|  |  |
| --- | --- |
| Dialog – Abstract (View) | |
| Responsibilities: | **Collaborators:** |
| Define return to menu button | WinDialog – implementation |
| Define all jlabels needed for the dialog | TieDialog – implementation |

# 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. RankTable
  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
  5. Main.java

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. FileManager (Abstract)
     1. FileProfile

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 with the Game logic to track the current state of the game (whose turn it is).

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 Packages in the 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.

## Tests

JUnits tests were not used for this project. All testing done for this project can be found in the following section: Test plan.

## Test plan

1. Ensure all GUI’s render correctly, with 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.
   4. Load game GUI with different settings to ensure that settings was working properly.
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 known untested functionality.

## Requirement satisfaction

|  |  |
| --- | --- |
| Defined Requirements | Test Plan |
| GUI | 1a, 2a, 2b, 2c |
| Text Processing | 1a |
| Animation and Graphics | 1a |
| Storage and Retrieval of Information | 6a, 6b, 6c, 7 |
| Editing and Configuring the Software Product | 2d |
| Specific Requirements |  |
| Swapping between player mode and AI mode | 1a |
| Visually appealing menu screen | 1a |
| Game board with updating graphics | 3a, 3b |
| User Profiles | 6a, 6b, 6c |

# 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 as well as standard comments. Java is obviously a very powerful language and this was a very good class to take (which an awesome professor, no doubt). This project was not the most complicated or creative of the bunch here and it hasn’t necessarily held up to the standards that we wanted to get it too, but it was a very interactive and educational experience working with new people and a new language. It was fun getting to talk with each other during team meetings and difficult working together towards the end of the semester, but in the end we’ve all learned a lot and hopefully can carry it with us into our futures.