Beernanza: A Beverage themed memory game in Java!

Drinking can cause memory loss. Or even worse, memory loss. -Anonymous

In Belgium, beer is as common as day and provides one of the oldest and most fundamental parts of its world-renowned cultural identity. Conversely, Belgium is also considered to be one of the best places in the world for a high-quality university education, and as demonstrated by the international reputation of institutes such as KU Leuven and Ghent University, among others, it certainly succeeds in both departments!

Now, what happens when you combine these two essential elements of Belgian culture? Beernanza may provide the answer to that! A tile-based memory-match game that is focussed on Belgian's proudest tradition of beer, Beernanza can provide hours of fun while playing homage to some of the finest breweries in the world, from Belgium and beyond!!

Basic Rules of Play

The basic premise of Beernanza is quite simple. At the beginning of a game, a board of anonymous beer logos is created at random, each hidden under the default card setting. The player is responsible for trying to match two of the same beer logos as quickly as possible, while trying to avoid finding any mismatches! However, the player can also add certain elements that make the game trickier and more complex. This could include increasing the number of cards on the board, which can accommodate up to 12 logo pairs (or 24 total cards), as well as increase the speed at which cards are shown and quickly re-hidden. Depending on how many beers you may have had during game play, these features could provide a real challenge! Lastly, in each game session there exists a special card called "Hair of the Dog", which, when matched to its partner, returns the player's score to 0. While this card could be useful early on to eliminate a negative score, if selected as the last card in the game, the player could undo an entire session of hard work with one move! Note that this card is only added to the board when the number of cards used is 8 or greater.

Beernanza also supports a two-player mode on a shared screen, with each player getting alternating turns and a unique score. For each session, there is the possibility to save the player's score (note that in two-player mode, only the highest score in saved). Finally, these scores can be seen at anytime from the main menu, and reset at anytime if the user (very carefully) considers it worthwhile.

A complete, comprehensive set of playing instructions and rules can be found through the "Click Here for Help" function provided on Beernanza's welcome screen.

Java Classes & Packages

Classes are listed alphabetically in their respective packages: Graphics, GUI, or Logic.

Graphics: *establishes the visual card board for the player to interact with during game sessions.*

- CardPanel – This class creates a panel that contains the cards for basic game play, each contained within a JButton that can be clicked and matched to another identical button. Images are read into string arrays from folders in the source directory, and populate the panel in pairs in a random configuration each time the panel is created.
- GamePanel – provides a simple class extending a JPanel to package the CardPanel into another JPanel before adding this component to the playing frame, improving visibility and layout of the final playing window.
- CardController - CardController extends a JFrame and provides the complete game play window for the player. In this window the card panel is visible, as well as two side panels that show current scores for either single or multi player, as well as the current turn for either player

one or two. Three buttons also provide the ability for the player to save current score, view past high scores, and exit the current game session.

GUI: *provides the visual components for a welcome screen, score tracking, game help, and in-game information. Allows for high scores to be saved and viewed, and the rules of play to be accessed.*

- GameHelp – A simple class to establish a JButton Action Listener to view the help page from the main welcome window, which will show the rules of play and basic game information.
- GameInformation - from the welcome screen, the player can retrieve basic game information and rules of play in a new frame from a JButton that extends GameHelp
- GameStarter - establishes JButton Action Listener to start game and connect to the card controller
- GameStartScreen - GameStartScreen provides the welcome screen for the player when the game is executed. It is a JFrame populated with various buttons, sliders, and check boxes to select game preferences before execution. From this screen the player sets the speed difficulty, board dimensions, single or multi player mode, and can select the alternate card deck. The player may also view the game help/rules of play window, high scores of past game sessions, and can reset the high score contents. Finally, the player may create a new game session.
- ScoreActivator - creates a JFrame to view the saved high scores. If no high scores exist in the save file, it will create a separate JFrame to specify this to the viewer.
- ScoreResetter - Links with JButton on the welcome screen to delete the contents of the game score file. A new game score CSV will automatically be generated when new scores are added and no game score file exists, achieved by the ScoreClicker class in the Logic package.
- ScoreViewer - Creates a new JFrame that is accessible from the welcome screen, that contains high scores from past players and includes relevant game information such as the user name, date of play, dimensions of the board, timer speed, and a difficulty setting generated by the DifficultyAssessment class (a string representing the overall difficulty of the player's session).

Logic: *Contains the classes to create the behaviour for the game board, cards, and score allocation, and game difficulty. Establishes card visibility through clicking and the ability to save game scores.*

- CardClicker: This class establishes the logic for when cards are clicked. Use of a timer dictates the amount of time that cards are visible before they are either matched, or reset. Two cards need to be viewed at a time to find a potential match. The system keeps track of the number of clicked images in order to evaluate scoring when matches occur. If the player selects the multi-player mode, the system tracks a string for player turn to make alternating turns possible.
- ChangeScorer – This class provides logic to generate scenarios for adding the score for the user during game play. Methods are written for both single player and multi player options, and write the score to a card panel that is visible in a text field during game play. Three scoring scenarios are possible, and system uses a duration class to track the amount of time passed between card matches, and incorporates this duration into score calculation.
- DifficultyAssessment: using a combination of the timer speed and number of cards selected, each session of the game where a score is saved gets a string representation of the game difficulty, saved to the high scores page and score CSV.
- ScoreClicker: This class adds the ability to save your score by clicking the button on the card frame, with the option to input your name. If high scores do not exist in the system, it will create a new CSV file. If scores already exist, it will append to the existing list in CSV format. For the two player setting, the game will automatically save the higher of the two when the prompt is selected

- TimerTracker - This class establishes the starting default position of the cards at the beginning of the game, where the cards will be face down with the starting face, and the timer will not be running until the first card click is executed by the player.

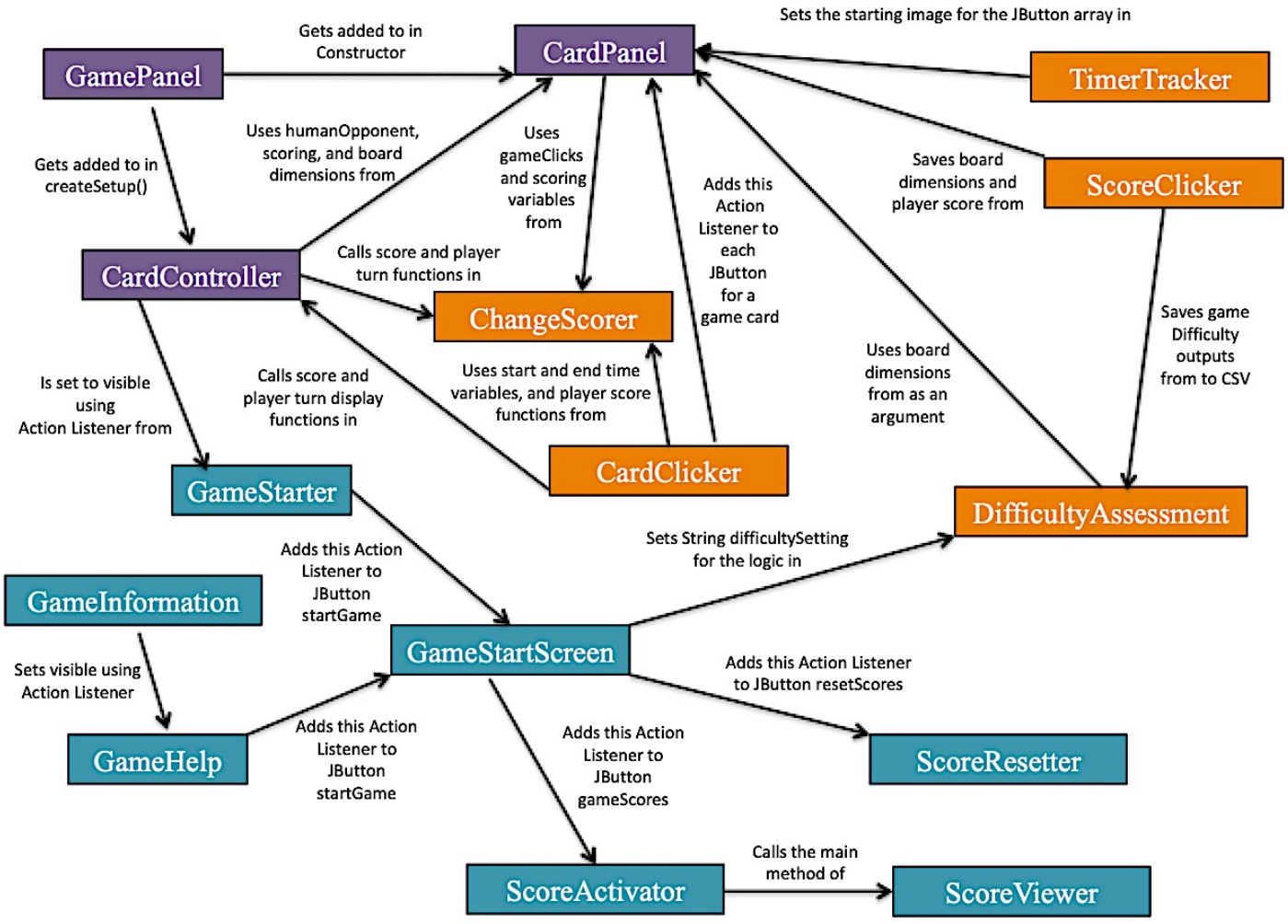

Figure 1. Diagram showing the relationships held among the classes contained with Beernanza. The direction of the arrow indicates the origin class of the variable or method calling, and the particular variable or method is provided. Classes are colour coded according to the following groups: package *Graphics* in purple, *GUI* in turquoise, and *Logic* in orange.

Strengths and Weaknesses

Strength: Separation of code leading to readability and code portability

Making use of three packages to separate classes based on their function within the program proved to be useful for tracking the specific purpose of various classes as well as identifying the variables shared among classes with similar functionality. By establishing packages related to graphical components, basic gameplay, and in-game behaviour from player clicking, it was easier to develop more complex scenarios for game behaviour and extend certain basic game principles into more complex and involved methods. This includes being able to create a sophisticated score tracking system that recorded variables from many different classes into the CSV file, which the user is able to access at anytime from either the in-game window or the welcome screen, as well as create the logic to alternate turns for the two-player mode so multiple players could engage in a game session. Furthermore, creating packages that bundled together functionality allowed for me to track the components involved in a certain part of game rendering during testing, which saved me time during

troubleshooting/debugging and made the logic of the game increasingly clear as I attempted to add more advanced elements to the game such as special cards, alternate sets of cards, in-game speed difficulty, etc.

Strength: Implementation of multiple scoring scenarios for both single and two-player modes

As a result of separating blocks of code into individual classes based on functionality, I believe that I was able to successfully come up with several creative game scoring scenarios for both single and two-layer modes in Beernanza. This included creating a scoring deficit for failing to match cards, as well as adding bonuses for both getting a match on the initial attempt of the game, and a bonus for back-to-back matches. This logic was accomplished using a combination of a integer variable that tracked game clicks, and a block of code that was able to reset this value to a targeted value based on in-game decisions made by the player. Finally, using a game timer to track the duration of time between matches, and applying this time to the player's score, I added an extra element of excitement and urgency to the game, as players will be aware of their potential score increase continuously becoming smaller as they attempt to find a new card match, and therefore may be more inclined to focus on the game positions in an attempt to get a higher score.

Weakness: Lack of a completely visually aesthetic and themed GUI

Despite having developed a unique theme in Belgian beer for Beernanza, I wasn't able to develop the exact GUI that I had envisioned to have for my final game. If given more time, I would have liked to have created a customized JProgressBar that appeared in the in-game window, which would have taken the shape of a beer bottle and filled up progressively as the number of cards matched increased. While I had investigated adding this component near the end of development, I had some difficulties in rendering the image that I had wanted for this final addition, so I decided to exclude it as it wouldn't have added much value to the in-game window, and may have proven to be more of a visual distraction for the player than a valuable graphical component. Additionally, I would add more font and image-based theming to the GUI components of Beernanza to make them look more authentic and stand out from other Swing-based Java games that use similar functionality in JFrames, JPanels, etc., adding to the player's overall experience when playing the game.

Implementation Difficulties, Next Steps & Improvements

Difficulty in implementing a robot class to simulate computer clicks for a computer opponent

Having been successful in creating a two-player mode for Beernanza, I was unable to finalize a fully functional computer-simulated mode where the player could play the game against a computer-simulated opponent. Initially, I had investigated use of a RobotClicker class that extended a RobotClicker function using a random generation of integers that would dictate the random on-screen positioning of the cursor. This class was intended to be able to click randomly on the array of cards in a specific mode where the system would be able to alternate turns between the player and computer, and could identify when the cursor would be available to the robot. However, in practice I had difficulty establishing consistent clicking on the card panel from the robot, and ended up having a robot simulator that clicked on many elements that were outside of the game window, resulting in accidental game window closing or triggering outside application events. If given more time, I would either investigate a more controlled allocation of the robot clicker on the screen, possibly using a tighter range of integers available to the clicker, or looking for a java class that could cause random action listeners to occur without the use of the cursor.

In terms of general next steps, I would continue to try to optimize the visual components of the game to make it increasingly visually appearing to the player, as well as try to establish the ability for players to import a unique collection of beer or beverage-themed cards and visual accessories so that the gameplay would continue to be exciting and fresh for the players after repeated game sessions.