**Explanation**

My game is a classic memory game. The aim is to find pairs of the same image to score points. You play by clicking on the (X) tiles and flipping them over. See rules.txt for exact rules. And special tile functionality.

**Improvements**

As my last project was incomplete, the majority of this submission is new. While I have kept the original game idea and Graphical interface, the underlying game logic is rebuilt mostly from scratch. In terms of the Project requirements, I have improved the project by implementing the following requirements into the current submission:

* Game Settings: Theme (uploading your own .pngs) and reset theme options; game modes: player vs player, player vs computer (+difficulty), and player vs self; rules; player names; high score table;
* Game Functionality: Algorithm for computer player; high score button beneath the game area; turn indicator; unique scoring system for computers; different score modifiers for Player 1 and Player 2; two special Tiles indicated by their border when flipped, smoother gameplay (removed Dialogue windows between terms).

In terms of project design, I attempted to make the code more modular by breaking existing code into different classes and making new classes for new features when possible instead of adding them onto existing classes. The packages, classes, and their respective responsibilities/functions are listed below.

**Packages and Classes**

# behaviour

* GameBehaviour
  + Runs a turn, returns state of game
  + Determines whether a turn was successful or unsuccessful and who the current turn belongs to
  + Controls score allocation and score modifiers
* GameMode (enum)
  + Defines existing game modes
* GameSettings
  + Stores game settings
* GameState
  + Saves the state of the game at any given moment
* GameTask
  + Thread task, for the game thread, runs the game logic

# cards

* Card
  + Defines Card as Graphic element with Game-logic attributes (flipped/unflipped, special/normal)
  + Defines methods to manipulate Card graphics and Card attributes
* SpecialCardBehaviour (enum)
  + Defines all Card behaviors as enumeration literals, special and normal
  + Defines methods to modify UI for special cards
  + Defines method to randomly allocate special behaviour to cards

# loaders

* ImageLoader
  + Defines Method to load images according to defined theme to use for cards’ front sides
  + Defines method to load card backside
* RuleLoader
  + Defines method to load rules from a given path

# players

* Player(interface)
  + Declares behaviour and attributes for a player
  + Requires subclasses to define Player-specific methods including turn behaviour and score modifier getters
* CheatingComputerPlayer
  + Player implementation that defines automatic computer-generated turns
  + Defines a cheat mechanic based on a cheat chance determined by difficulty
* DumbComputerPlayer
  + “”
* HumanPlayer
  + Player implementation that awaits human interaction

# scoring

* Score
  + Defines a row in the high score table
* ScoreKeeper
  + Defines method to store high score information in external file for persistent storage

# ui

* GameFrame
  + Defines and executes methods to create and manipulate the Game window and child elements
  + Defines methods for retrieving Card List on Game Window and checking if Unflipped cards remain on board
  + Runs thread for UI
* GameUI (interface)
  + Defines behaviour for interaction between **GameBehaviour** and the UI.
* GameUIImpl
  + GameUI Implementation for working with **GameFrame**
* MainFrame
  + Defines and controls the start window
* HighscoreActionListener
  + Action listener definition to open the high score table dialogue
* CardClickListener
  + Action listener implementation for listening to card clicks on the gameArea and alerting awaiting elements of the event

**Changing Theme**

I approached requirement of changing the theme of the game by trying to allow users to select a folder of their own to upload their own icons to match the game with. The implementation is fairly straight forward. I used the image loader to define a method of actually loading a folder and retrieving images inside. Then, I allowed the location of the file that is retrieved from to be decided by the player via the “Change Theme” button on the start window, else the images will be loaded from a default path.

**Strengths**

* + Extensive use of methods increases code reusability
  + Simplicity of game behaviour and UI
  + Encapsulation
    - Variables are sent/received through getters and setters
  + Having many classes makes the game more modular and changeable in the future.
    - E.g. a new GameMode could be added easily by creating a new enumeration literal.

**Weaknesses**

* GameBehaviour is very rigidly designed for two players. Could not easily include more/less players.
  + Example: I tried to implement a Player-vs-self mode, yet it still has to have two players instances.
* Issues between UI and logic
  + I tried to achieve a good separation between UI and Game-logic, attempting to implement [different threads](https://beginnersbook.com/2013/03/multithreading-in-java/) for both, however I am unsure if this implementation is correct or desirable.
* Computer player algorithm does not imitate human player. I considered trying to create a memory list from which the computer could try to “remember” where the cards are and increase this memory for difficulty, however this proved to be too problematic to be able to implement.
* Some classes contain more code/functionality than intended (e.g. GameFrame). I could have tried to split up some of the code here to make it more manageable (e.g. the single-use listeners could have their own class, helper methods for managing cards could have their own class.)

**Diagram**

Below I have attached an attempted UML Class Diagram of the entirety of the project. Although through trying to include all classes and relationships the diagram has become very cluttered, it is able to clearly show the distinction of the classes in their respective packages.

A close up of a map

Description automatically generated