Dobble: A Complex Cognitive Modelling Approach*

Linas Raicinskis s5402212
Kyriakos Antoniou s5715881
Pedro Lavin s5509025

August 29, 2025

1 App Overview

Dobble is a very popular board game that can be played by many people, but mainly between two people. There is a deck that contains several cards, each card with 8 different images in it. The images and the amount of these are designed so that every card of the deck shares exactly one common image between each other. In each of the turns of the game, each player draws a random card from the deck and has to find the matching image as fast as possible, before the other player. Once any of the players have found it, they win one point and both cards get shuffled. The game goes on until there are no more cards in the deck, and the player with the highest number of points becomes the winner. In our App, we implement the game of Dobble, with slight changes, especially regarding the punctuation, and adding penalties from wrongly chosen images.

2 Implementation

2.1 Code Structure

The code of the app relies on the interaction between several classes. This section gives a description of each class and an explanation of the role each of them plays in the app.
The 6 classes are *Card*, *CardPair*, *Deck*, *DobbleGame*, *ImageProperties* and *Player*.

Card

The Card class is the basic element of the game. Each player has one card, and each card has 8 symbols. The class has a variable for the list of images, and two more arrays which define the rotation and size changes of the corresponding images. The method *Contains* takes an integer as input and returns true if the card contains the image corresponding to that integer. The rest of the methods include setter and getter for the rotation and size arrays.

CardPair

The CardPair class is composed of two card objects. The principal purpose of this class is to check whether a certain image is contained in both cards simultaneously. This is done with the method *isMatchingSymbol*, which takes an integer corresponding to a specific image as a parameter and returns true if it is present in both cards.

Deck

The Deck class contains the generateCards method, which is responsible for creating every possible card with 8 images so that there is only one common image between any two cards. It first generates random combinations of images and then iterates through every possible image, discarding those cards that share more than 1 image. Finally, it shuffles the cards with the *ShuffleCards* method, which essentially assigns a random order to the generated cards.

*w

DobbleGame

The DobbleGame class controls the dynamics of change of the cards. When a game turn is finished (when one of the two players finds the common object between the cards), the methods *updateCardPair* is called and two new random cards are selected.

ImageProperties

The ImageProperties class is very basic and defines the different colours of each image, so that they can be retrieved when the mode looks at each image and saves their properties to memory. These properties are then compared between each other to check whether it has found the same image.

Player

The Player class keeps track of every feature of the player, such as the name and score. When the player successfully clicks on the correct image a point is added to its score, and when it chooses an incorrect image, a point is deducted from the score.

Game Activity

This section explains the logic followed by the game, from when it starts until each of the players makes a move. When the "Start" button is pressed, a 4-second counter appears at the right of the screen to indicate that the game is about to start. Then, two cards are randomly picked from the deck and presented to the players, each with 8 images which are pressable buttons. In this part of the code the user interface is built and it is kept up to date after the players press different buttons. If a player presses a wrong image, one point will be deducted from their score. If a player correctly finds the matching image, one point will be added to their score, and new cards will be drawn and thus shown in the interface. In this part of the code, the game logic is also checked. This refers to whether the image pressed actually corresponds to the matching image. Additionally, this part is responsible for highlighting in a red-ish outline every image that the model is looking at.

2.2 User Interface

The design of the user interface is composed of 3 stages. The first one appears just at the beginning of the game, with no images on the players' cards and both scores set to 0. A countdown from 4 seconds starts ticking, displayed at the right of the screen. This is the time for preparation for every player before the game actually begins. The second stage and the third repeat as long as the game goes on. After the preparation stage, the two cards are displayed, one for each player. The top card belongs to the model's card, while the bottom one is the player's card. The images in the card are presented with a random degree of rotation and size scaling, arranged in a square around an empty space. In the case of the model, inside of the empty space, there is a number that indicates the milliseconds it will take to find the correct image. As the model looks at each image to see its attributes, a red outline surrounds that specific image, so that the human player can have a sense of what the model is looking at. When either of the players wins, a message with the words: "Congratulations". If they lose, the message: "Incorrect, ". After someone has won, the screen freezes and the 4-second countdown begins and when it reaches 0, the cards are updated. These 3 stages of the UI are shown in figure 1

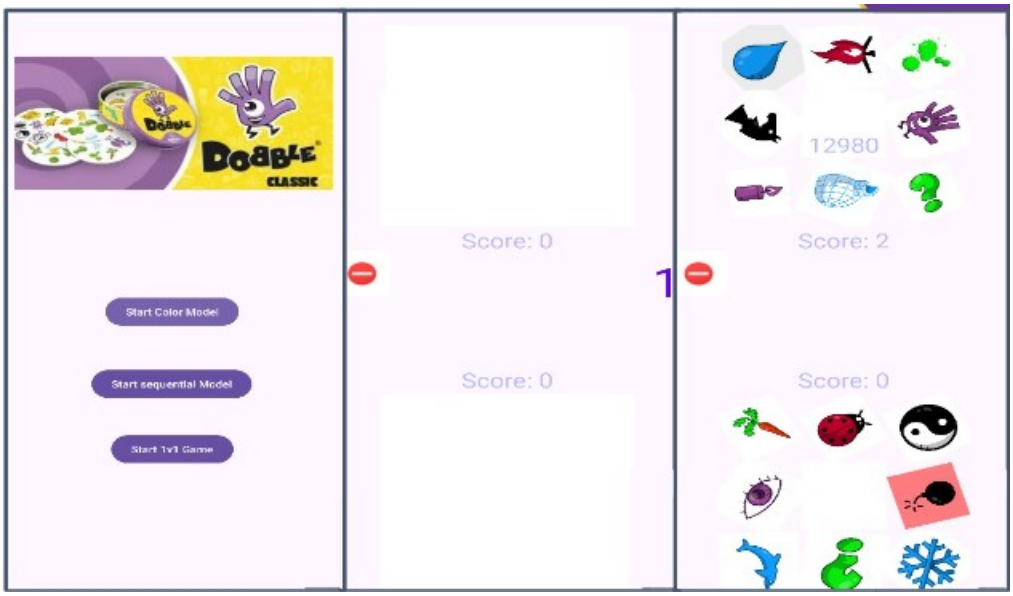

Figure 1: Images showing the 3 user interface screens shown in the game.

2.3 Model

The most essential thing that we needed to simulate was the eye movement of the model. This is, how much time it would take for it to move from image to image, as this is the main component in the total time in which the model finds iterates through all of the images and finds the correct one. Many research papers have focused on the speed of these movements,and have determined that eye movement is mainly determined by two phenomena, fixations and saccades [Kom13]. Fixation is the point in which the eye is approximately stationary and receives all visual input. Saccades is the movement that occurs between fixations, where the gaze is shifted from input to input. The fixation duration is in general about 250-300ms, but can fall down to 20-30ms[Nah22]. Following this data, we implemented the model so that the time it takes to move between close-by images (in the same card) was 85ms, while the time it takes to go from card to card was set to be 300ms. In order to take into account the saccade, we added a linear increase on the duration time depending on the image rotation and scaling factor. This was done as rotated images are harder to perceive than normal ones [Sal80]. The rotation factor was set so that if an image was completely rotated (180 degrees), that would cost the model an extra 25ms.

Sequential

In the Sequential option of the model, the most simple and easy-to-follow implementation is carried out. Essentially, the model looks one image at the top and then compares it with all of the images from the bottom card, following the same order, from left to right, from top to bottom. Once it has compared the first image with every single image in the bottom card without finding any match, it moves on to the second image of the top card and the process is repeated until a match between any two images is found. The sequential nature of this process makes it very tedious and slow, as the model has to look back and forth multiple times, even looking at many times at the same image. There is no priority of any kind between images, so it is a less realistic model. On the other hand, it is very efficient, as it records every image to memory and thus guarantees that it will find the corresponding match in a given amount of time.

Color Based

With this option, the model adopts a new strategy to find the matching image between cards. First, it looks at the top card images and finds what the most common predominant colour in them is. Once this is done, it checks whether there is any colour match in the bottom card. If at least one image shares the same colour, then the model looks at every image with that colour. If it does not find the

matching image, then the process is repeated with a less dominant colour until a match is found. The main advantage of this strategy is that it is much faster than the previous one, as there is much less back and forth, so the time is optimized and the match is found much faster. A disadvantage, although it really just makes the model more human-like, is that it is prone to more errors, as the images are stored in memory at once.

3 Evaluation

The evaluation of the game was done by presenting it to several of our friends, and asking them to provide as much feedback as possible, both positive and negative.

Positive Feedback

The main positive critic that the game obtained was how easy it was to play it. It was clear what the objective of the game was from the beginning, without further need of displaying instructions on how to play or anything similar. Additionally, another popular comment was that it was challenging to actually beat the model. This means that the model's level at playing of the game can be compared to that of a skilled human.

3.0.1 Negative Feedback

After hearing the criticism of some of the aspects of the game, some of the changes proposed were included later in the game. By the time the evaluation took place, there was no 1v1 mode included in the game. We included just by adding an option where no model played, and the buttons could only be pressed by mechanical motion. It was also suggested to change the interface, as it did not appear very user-friendly. To respond to this, we introduced the first screen to introduce the game in a more comfortable way, and highlighted those images the model was looking at so that the human player could have an idea of what they were facing. Another critic was that the punishment for incorrect guess might be a bit harsh. No action was taken for this as we considered it was part of the challenge of the game.

4 Conclusion

The game Dobble was reproduced using Android Studio and several classes to account for all the aspects of the game. As an addition to the game rules, we introduced the penalty for incorrect guessing. The game is originally designed to be played against a model, although also includes the option to be played between two people. The interface was created so that it was similar to the original game, regarding the images. The model was built using references from research about eye movement duration and fixation time to make it as realistic as possible. The model's performance was efficient when it came to finding the correct image at every turn, and the time it takes to do so can be regarded as reasonable. For future improvements, a more detailed strategy could be implemented, looking at secondary colours or the shapes of the images, which would make the model less dependent on detecting the colours.

References

[Kom13] Oleg Komogortsec. Automated classification and scoring of smooth pursuit eye movements in the presence of fixations and saccades. *SpringerLink*, 2013.

[Nah22] Banuka Nahanama. Eye movement and pupil measures: A review. *Frontiers of Computer Science*, 2022.

[Sal80] Timothee Salthouse. Determinants of eye-fixation duration. *American Psychological Association*, 1980.