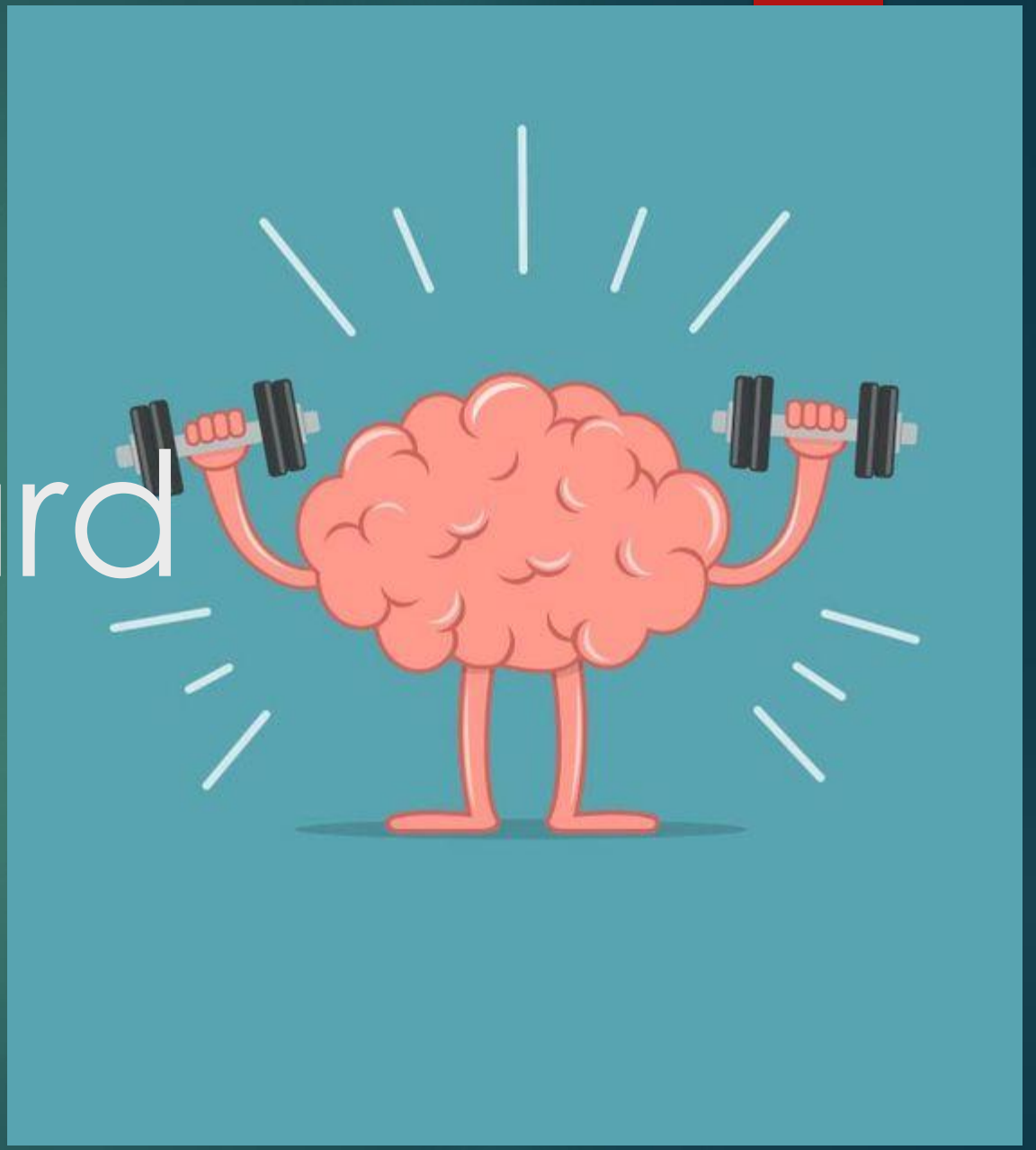
Memory Card Game



VAK: OBJECT ORIENTEERD PROGRAMEREN

GROEPSLEDEN: THAKOERDAT SHRIJA. 20230110

SOKROMO CYNTHIENE. 20230010

JACOTT JAMIRO. 20220203

Inhoudsopgave

1 Waarom Memory Card Game?

- We hebben gekozen om een eenvoudige, aantrekkelijke en uitdagende memory card game te programmeren, na onderzoek van verschillende game-ideeën.

1 Waarom Memory Card Game?

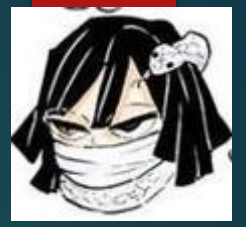
- We hebben gekozen om een eenvoudige, aantrekkelijke en uitdagende memory card game te programmeren, na onderzoek van verschillende game-ideeën.

2 Regels Memory Card Game

•Het spel begint met alle kaarten omgekeerd.
•Spelers draaien twee kaarten tegelijk om door te klikken.
•Kaarten worden vergeleken op paren.
•Bij een paar blijven ze zichtbaar en worden ze verwijderd; de speler krijgt 1 punt.
•Geen paar? Kaarten worden weer omgekeerd en de beurt gaat naar de volgende speler.
•Het spel eindigt als alle paren zijn gematched.
•Winnaar is de speler met de meeste punten.

3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.



Indeling van de code (header 1)

- WELCOME_PART_H

```cpp
#ifndef WELCOME_PART_H
#define WELCOME_PART_H

#include <iostream> // Voor invoer/uitvoer operaties
#include <Windows.h> // Voor Windows-specifieke functies zoals system() en Sleep()

// Function declarations
void setupConsole(); // Verklaring van de functie setupConsole()
void displayWelcomeScreen(); // Verklaring van de functie displayWelcomeScreen()
void displayProgressBar(); // Verklaring van de functie displayProgressBar()

// Function definitions
inline void setupConsole() {
    // Stel de tekstkleur van de console in op wit (0F)
    system("COLOR 0F"); // Wijzigt de console tekstkleur naar wit

    // Wis het scherm van de console
    system("cls"); // Verwijdert alle tekst van de console
}

inline void displayWelcomeScreen() {
    // Print de titel van het spel
    printf("\Memory Card game"); // Drukt de titel "Memory Card game" af in de console

    // Stel de codepagina in op 437 voor correcte ASCII-tekenweergave
    SetConsoleCP(437); // Wijzigt de codepagina van de console naar 437 (IBM PC)

    // Stel de uitvoer codepagina van de console in op 437 voor correcte weergave van ASCII-tekens
    SetConsoleOutputCP(437); // Wijzigt de uitvoer codepagina van de console naar 437 (IBM PC)

    // Toon het welkomstbericht en de namen van de spelers
    std::cout << "\n\n\n\n\n\t\t\t\t\tWELCOME"; // Toont "WELCOME" op het scherm met voldoende afstand boven en links
    std::cout << "\n\n\n\n\t\t\t\t PLAYER 1      PLAYER 2 "; // Toont de spelersnamen met afstand erboven
}

inline void displayProgressBar() {
    int bar1 = 176, bar2 = 219; // Definieert de karakters voor de lege en gevulde voortgangsbalk

    std::cout << "\n\n\n\t\t\t\t"; // Voegt afstand toe voor de voortgangsbalk

    // Toon de lege voortgangsbalk
    for (int i = 0; i < 25; i++) {
        std::cout << (char)bar1; // Print teken 176 (lege balk) 25 keer
    }

    std::cout << "\r"; // Zet de cursor terug naar het begin van de regel
    std::cout << "\t\t\t\t"; // Voegt afstand toe

    // Animeer het vullen van de voortgangsbalk
    for (int i = 0; i < 25; i++) {
        std::cout << (char)bar2; // Print teken 219 (gevulde balk) 25 keer
        Sleep(150); // Wacht 150 milliseconden tussen elke stap
    }

    std::cout << "\n\t\t\t\t" << (char)1 << "!"; // Print een uitroepteken na de voortgangsbalk
}

#endif // WELCOME_PART_H
```

3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.

Indeling van de code (header 2)

- PLAYER_PART_H

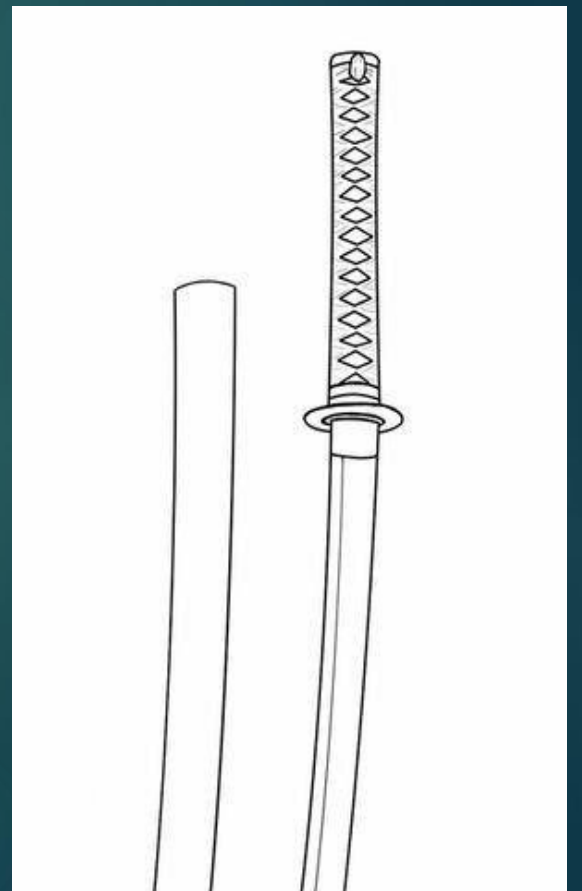
```
#ifndef PLAYER_PART_H
#define PLAYER_PART_H

#include <string>
using namespace std;

class Player {
public:
    // Constructor to initialize player's name and score
    Player(const string& playerName) : name(playerName), score(0) {}

    // Getter for player's name
    string getName() const { return name; }

    // Getter for player's score
    int getScore() const { return score; }

    // Increment player's score
    void updateScore() { score++; }

    // Switch current player between player1 and player2
    void switchPlayer(Player*& currentPlayer, Player& player1, Player& player2) {
        currentPlayer = (currentPlayer == &player1) ? &player1 : &player2;
    }

private:
    string name; // Player's name
    int score;   // Player's score
};

#endif // PLAYER_PART_H
```

3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.

Indeling van de code Libaries

- #include <SFML/Graphics.hpp>// deze libary dient voor het omgaan van graphics
- #include <Windows.h>// deze libary wordt gebruik voor specifieke windows API functies
- #include <vector>// heeft vector containers uit de STL
- #include <string>// gebruik van de stringklasse in STL (gebruiken voor tekst manupilatie)
- #include <iostream>// input en output libary
- #include <algorithm>// verzameling algoritmen idg wordt het gebruikt voor soorten zoeken
- #include <random>// deze dient om de kaarten te randomizen
- #include <ctime>// tijdfuncties die worden gebruikt voor getallengenerator
- #include "WELCOME_PART.h" //hier implementeer je de welocme stuk in de mainfile
- #include "PLAYER_PART.h" //hier implenteer je de player header file in je mainfile



3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.

Indeling van de code (Class 1)

- Class Card

```
16  class Card {
17  public:
18      Card(float width, float height) : shape(sf::Vector2f(width, height)), flipped(false), frontTexture(nullptr) {}
19      // constructor met parameters, shape van de kaart, de kaart is niet omgedraait en fronttexture is nog niet ingesteld.
20      void setPosition(float x, float y) {
21          shape.setPosition(x, y);
22      }// positie van de kaarten instellen
23      bool contains(sf::Vector2f point) const {
24          return shape.getGlobalBounds().contains(point);
25      }//controleert of de gegeven punt binnen de grenzen van de kaart ligt.
26      void draw(sf::RenderWindow& window) const {
27          window.draw(shape);
28      }// tekenen van de kaart in  de venster.
29      void flip() {
30          flipped = !flipped;
31          shape.setTexture(flipped ? frontTexture : backTexture);
32      } // dient voor het draaien van de kaart. indien het gedraait is zal het fronttexture weergegeven worden
33      bool isFlipped() const {
34          return flipped;
35      }// dient voor het terugkeren van de kaart
36      void setFrontTexture(const sf::Texture& texture) {
37          frontTexture = &texture;
38          if (flipped) shape.setTexture(frontTexture);
39      }// dient voor het stellen van de voorzijde (fronttexture)
40      void setBackTexture(const sf::Texture& texture) {
41          backTexture = &texture;
42          if (!flipped) shape.setTexture(backTexture);
43      }// dient voor het stellen van de achterzijde (backtexture)
44      static void loadTextures(const vector<string>& filenames, vector<sf::Texture>& frontTextures, vector<int>& cardIndices) {
45          for (const auto& filename : filenames) {
46              sf::Texture texture;
47              if (!texture.loadFromFile(filename)) {
48                  cout << "Error loading texture: " << filename << endl;
49                  return;
50              }
51              frontTextures.push_back(texture);
```

```
52          }
53          cardIndices = { 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8 };
54          shuffle(cardIndices.begin(), cardIndices.end(), default_random_engine(static_cast<unsigned int>(time(0))));
55      }//Deze methode laadt een reeks afbeeldingen (textures) voor de voorkant van de kaarten,
56      //en shufflet vervolgens de indices om de kaarten willekeurig te verdelen.
57      static void setCardPositions(vector<Card>& cards, float cardWidth, float cardHeight, float spacing) {
58          for (int i = 0; i < cards.size(); ++i) {
59              int row = i / 6;
60              int col = i % 6;
61              cards[i].setPosition(col * (spacing + cardWidth + spacing), row * (spacing + cardHeight + spacing));
62          }
63      }// deze functie dient voor het positie van de kaarten in de venster
64
65  private:
66      sf::RectangleShape shape;// grafische representie van de kaart. hoe de kaart eruit zal zien
67      bool flipped;// geeft aan of de kaart is omgedraaid
68      const sf::Texture* frontTexture; //pointer naar de voorkant van de kaart
69      const sf::Texture* backTexture;// pointer naar de achterkant van de kaart
70  };
71
```

3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.

Indeling van de code (Class 2)

- Class MemoryGame

```
class MemoryGame {
public:
    MemoryGame();//constructor, initialiseert het venster met de daaropbestemde functies
    void run();//Start de hoofdgameloop waar het spel wordt uitgevoerd

private:
    void draw();//Tekent alle kaarten en bijgewerkte scores
    void handleEvents();// verwerkt de handelingen van de gebruiker(muisklick)
    void checkMatch();//controleren van de omgedraaide kaarten (gelijksoorten)
    void resetGame();// restart de game na 1 ronde

    sf::RenderWindow window;//sfml venster plaatsen op de scherm
    vector<Card> cards;// dient voor de kaarten in het spel
    vector<sf::Texture> frontTextures;//textures voorkant van de kaarten
    sf::Texture backTexture;// textures achterkant van de kaarten
    vector<int> cardIndices;//een geshuffel lijst van indices om de kaarten willekeurig te verdelen
    const vector<string> filenames = {
        "C:/Users/User/Desktop/Images/1.jpeg",
              "C:/Users/User/Desktop/Images/3.jpeg",
               "C:/Users/User/Desktop/Images/2.jpeg",
                "C:/Users/User/Desktop/Images/4.jpeg",
                 "C:/Users/User/Desktop/Images/5.jpeg",
                  "C:/Users/User/Desktop/Images/6.jpeg",
                   "C:/Users/User/Desktop/Images/7.jpeg",
                    "C:/Users/User/Desktop/Images/8.jpeg",
                     "C:/Users/User/Desktop/Images/9.jpeg",
    };
    const std::string backFilename = "C:/Users/User/Desktop/Images/BACKGROUND.jpeg";
    //dient voor de texture van de achterkant van de kaarten
    const float cardWidth = 100.0f;
    const float cardHeight = 150.0f;
    const float spacing = 20.0f;
    // lengte, breedte en verte van de kaarten
    int firstCardIndex;// dient voor de omgedraaide kaarten
    int secondCardIndex;
    bool checkingMatch;//aangeeft of het spel wacht op matchcontrole
    sf::Clock clock;// dient voor de tijd tussen het beurt van de volgende speler
```

```
110     Player player1;//Objecten en pointer voor spelersbeheer
111     Player player2;
112     Player* currentPlayer;
113
114     sf::Font font;//dient voor het behouden van de score (scoreboard)
115     sf::Text scoreText;
116     sf::Text turnText;
117 };
118
119 MemoryGame::MemoryGame()
120     : window(sf::VideoMode(800, 600), "Memory Game"), firstCardIndex(-1), secondCardIndex(-1), checkingMatch(false),
121     player1("Player 1"), player2("Player 2"), currentPlayer(&player1) {
122     Card::loadTextures(filenames, frontTextures, cardIndices);
123     //window en dient voor het bijhouden van de geselecteerde kaarten
124     if (!backTexture.loadFromFile(backFilename)) {
125         cerr << "Error loading back texture: " << backFilename << endl;
126     }//dient voor de backtexture. indien het mislukt het functie verder
127
128     for (int i = 0; i < 18; ++i) {
129         cards.emplace_back(cardWidth, cardHeight);
130         cards[i].setFrontTexture(frontTextures[cardIndices[i]]);
131         cards[i].setBackTexture(backTexture);
132     }//instelling 18 kaarten, back en front texture
133     Card::setCardPositions(cards, cardWidth, cardHeight, spacing);
134     // nadat alles wordt ingesteld worden de kaarten in de posities gezet
135     if (!font.loadFromFile("C:\\Windows\\Fonts\\Arial.ttf")) {
136         cerr << "Error loading font!" << endl;
137     }// deze file dient voor de lettertype van de tekst
138
139     scoreText.setFont(font);
140     scoreText.setCharacterSize(24);
141     scoreText.setFillColor(sf::Color::White);
142     scoreText.setPosition(10, 550);
143     // instelling scoretext
144     turnText.setFont(font);
145     turnText.setCharacterSize(24);
146     turnText.setFillColor(sf::Color::White);
```

3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.

Indeling van de code (Class 2)

- Class MemoryGame

```
146     turnText.setFillColor(sf::Color::White);
147     turnText.setPosition(550, 550);
148 }//instelling wie de volgende spelers beurt is text
149
150 void MemoryGame::run() {
151     while (window.isOpen()) {//game blijft draaien zolang het venster open is
152         handleEvents();// verwerking van de gebruikers invoering
153         if (checkingMatch) {// controleren van de kaarten en tijdsvertraging
154             if (clock.getElapsedTime().asSeconds() > 1.0f) {
155                 checkMatch();
156             }
157         }
158         window.clear();// wissen van de window om een nieuw game te starten
159         draw();// opnieuw de kaarten en gegevens plaatsen in de window
160         window.display();// het inhoud plaatesen
161     }
162 }
163
164 void MemoryGame::draw() {
165     for (const auto& card : cards) {//&auto(constante referentie)
166         card.draw(window);
167     }
168
169     scoreText.setString(player1.getName() + ": " + to_string(player1.getScore()) + "  " +
170         player2.getName() + ": " + to_string(player2.getScore()));//namen en scores van beide spelers op te nemen
171     turnText.setString(currentPlayer->getName() + "'s Turn");//dit wordt gedaan door getname en getScore
172     //aangeving beurt
173     window.draw(scoreText);// tekent het scoreTekst op de venster(window)
174     window.draw(turnText);//tekent het scoreTekst op de venster(window)
175 }
176
177 void MemoryGame::handleEvents() {
178     sf::Event event;//wordt aangemaakt om individuele gebeurtenissen op te slaan
179     //die door het venster worden gegenereerd
180     while (window.pollEvent(event)) {//alle gebeurtenissen uit de gebeurteniswachtrij
181         //en verwerkt ze een voor een totdat er geen meer zijn
182         if (event.type == sf::Event::Closed)
```

```
182         if (event.type == sf::Event::Closed)
183             window.close();//venster gesloten indien de gebruiker dat doet
184         if (event.type == sf::Event::MouseButtonPressed && !checkingMatch) {//uisklik is gedetecteerd en er geen controle op een match wordt uitgevoerd
185             //wordt verder gekeken naar het type muisklik.
186             if (event.mouseButton.button == sf::Mouse::Left) {//controleert linkermuisknop
187                 sf::Vector2i mousePos = sf::Mouse::getPosition(window);//huidige positie van de muis in vensterco[illegible]rdinaten.
188                 for (int i = 0; i < 18; ++i) {//reageert op de klikken van de muis
189                     if (cards[i].contains(static_cast<sf::Vector2f>(mousePos)) && !cards[i].isFlipped()) {
190                         // de muispositie zich binnen de grenzen van de kaart
191                             //bevindt en of de kaart niet al omgedraaid is
192                         if (firstCardIndex == -1) {// controleert of index is ingesteld, indien niet wordt het wel ingesteld
193                             firstCardIndex = i;
194                         }
195                         else if (secondCardIndex == -1) {
196                             secondCardIndex = i;
197                             checkingMatch = true;
198                             clock.restart();//vlag wordt geactiveerd, en de klok wordt gereset
199                         }
200                         cards[i].flip();//roept de functie flip aan
201                         break;//beeindiging van de forloop
202                     }
203                 }
204             }
205         }
206     }
207 }
208
209 void MemoryGame::checkMatch() {
210     if (cardIndices[firstCardIndex] == cardIndices[secondCardIndex]) {// dient voor het checken of
211         //de kaarten hetzelfde zijn
212         currentPlayer->updateScore();//update van de score indien de kaarten overeenkomen
213     }
214     else {// indien ze niet overeenkomen zullen de kaarten weer omgedraaid worden
215         cards[firstCardIndex].flip();
216         cards[secondCardIndex].flip();
217         currentPlayer->switchPlayer(currentPlayer, player1, player2);//player wordt dan verwisselt
```

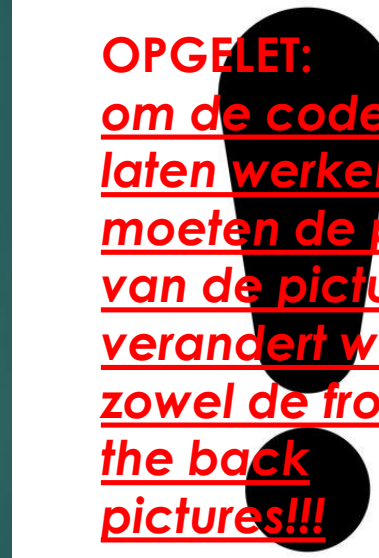
3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.

Indeling van de code (Class 2)

- Class MemoryGame

```
217             currentPlayer->switchPlayer(currentPlayer, player1, player2);//player wordt dan verwisselt
218         }
219         firstCardIndex = -1;
220         secondCardIndex = -1;//deze functies worden gereset
221         checkingMatch = false;
222 
223         bool allFlipped = all_of(cards.begin(), cards.end(), [](const Card& card) { return card.isFlipped(); });
224         if (allFlipped) {//controle of alle kaarten omgedraaid zijn
225             cout << "Game over! " << (player1.getScore() > player2.getScore() ? player1.getName() : player2.getName()) << " wins!" << endl;
226             resetGame();// reset wordt aangeroepen
227         }
228     }
229 
230     void MemoryGame::resetGame() {
231         shuffle(cardIndices.begin(), cardIndices.end(), default_random_engine(static_cast<unsigned int>(time(0))));
232         //shuffelen van de kaarten
233         for (int i = 0; i < cards.size(); ++i) {//herstellen van de kaarten
234             cards[i].setFrontTexture(frontTextures[cardIndices[i]]);
235             if (cards[i].isFlipped()) cards[i].flip();// indien alle kaarten omgedraaid zijn zal het game eindigen
236         }
237         player1 = Player("Player 1");// opnieuw intitialiseren van de players
238         player2 = Player("Player 2");
239         currentPlayer = &player1;// player 1 zal dan opnieuw moeten spelen
240     }
241 
```



OPGELET:
om de code te laten werken moeten de path van de pictures verandert worden zowel de front as the back pictures!!!

3 Inhoud van de code.

Deze code bestaat uit 2 headers(waarvan 1 een class heeft), 2 Classes en 1 Main Function in de main code.

Indeling van de code

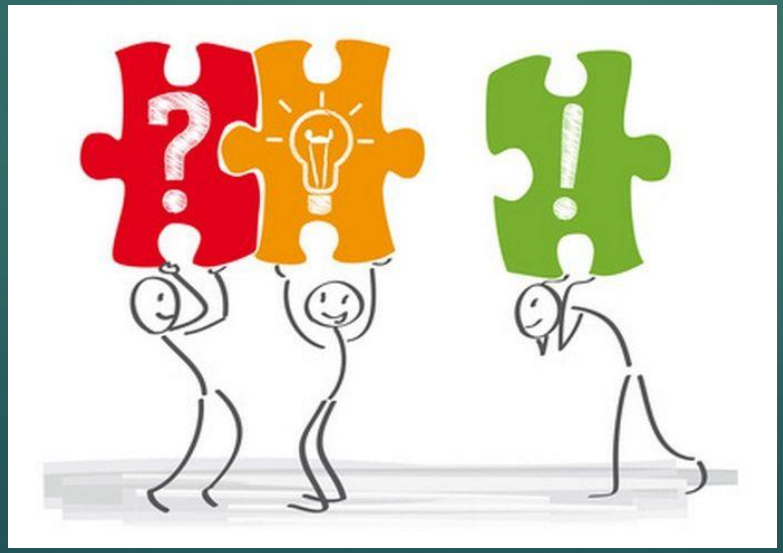
- Main Function

```
245  int main() {
246
247      displayWelcomeScreen();
248      displayProgressBar();
249      system("Pause");
250      // maakt 2 players
251      Player player1("Player 1");
252      Player player2("Player 2");
253
254      // Set the current player to player1
255      Player* currentPlayer = &player1; // plaatst de current player naar player 1 automatisch
256
257      // switchen van players
258      currentPlayer->switchPlayer(currentPlayer, player1, player2);
259      cout << "Current Player: " << currentPlayer->getName() << endl;
260
261      MemoryGame game; // Creëert een instantie van de MemoryGame klasse
262      game.run();      // Start de hoofdgame
263      return 0;
264  }
265
266  // Beëindigt het programma succesvol
267
```

4 Demon-stratie Code

5 Conclusie

- Het maken van de memory card game was erg leerzaam. Ondanks problemen met samenwerking en tijdsdruk, hebben we het leuk gevonden om een simpel maar uitdagend spel te ontwikkelen. We hebben onze programmeervaardigheden in SFML en C++ verbeterd.



EINDE!!!!

- Bedank voor het luisteren

Vragen??