Projektdokumentation Modul 326

|  |  |  |
| --- | --- | --- |
| Auftraggeber: |  | D. Fahrni |
| Autor\*in: |  | Davide Di Giovanni [digiovannid@bzz.ch](mailto:digiovannid@bzz.ch)  Tenzin Dehukar  dehukard@bzz.ch  Saam Eymany  eymanys@bzz.ch |
| Klasse: |  | IM19b |
| Projekttitel: |  | Memory |

Inhaltsverzeichnis

[Projektbeschreibung 2](#_Toc38719552)

[Anwendungsentwurf 3](#_Toc38719553)

[Testfälle 11](#_Toc38719554)

[Lösungsbeschreibung 20](#_Toc38719555)

[Anhang 21](#_Toc38719556)

Abbildungsverzeichnis

[Abbildung 1: Muster **Fehler! Textmarke nicht definiert.**](#_Toc1036253)

Tabellenverzeichnis

[Tabelle 1: **Fehler! Textmarke nicht definiert.**](#_Toc38719557)

[Tabelle 2: **Fehler! Textmarke nicht definiert.**](#_Toc38719558)

# Projektbeschreibung

Die Vorgabe ist ein Memoryprogramm zu programmieren. Das Memoryspiel, bestehend aus Bilderkarten, kann von zwei Spielern gespielt werden. Diese zwei Spieler können Punkte sammeln, indem sie die gleichen Karten ziehen. Die Punkte werden einzeln für jeden Spieler gezählt. Am Ende werden die Gesamtpunktzahlen der Spieler verglichen und der mit einer höheren Punktzahl als Gewinner angezeigt.

Meine Vorstellung: Ich stelle mir vor, dass der Benutzer das Programm mit einem Knopf startet. Oben dran wird das Spiel Logo eingeblendet. Danach kommt die Auswahl von der Spielgrösse, die Möglichkeiten von 5x5 bis 10x10 Felder anbietet. Beim Druck auf die Spielgrösse startet das Spiel. Wenn man Karten richtig errät, werden diese ausgeblendet, falls nicht werden sie zurückgedreht. Man kann auch mehrere Spiele spielen, das heisst am Ende kann man wieder neustarten oder man beendet das Programm. Es sollte auch ein Endbutton geben, der nach jedem beendeten Spiel beim Betätigen das Spiel beendet und die Punktzahl ausgibt. Die Punktzahlen für die Spieler fangen bei jedem Spiel von null an, doch im Hintergrund werden die ganzen Punkte zusammengezählt und für die Statistik vorbereitet. Dabei wird die Statistik ausgegeben und die Daten in ein File gespeichert (Dateipfad wird angezeigt).

## Ziele

***Muss-Zielen***

***1.Ziel***

*Ein Layout suchen, welches zum Memory spiel passt. Dieses sollte aber sehr einfach gehalten werden und eher mit Farben beschmückt werden, welche die Aufmerksamkeit zum Spiel lenken und Benutzerfreundlich ist. Dieses Layout werde ich vor der Implementierung bis am 10.6.21 dokumentiert haben.*

***2. Ziel:***

*Der Kunde wünscht sich entweder Bilder von Autos oder Tieren auf die Memorykarten. Diese müssen bis am 9.06.21 vorhanden sein und von mir rasch im Programm eingefügt werden. Die Bilder müssen eine hohe Qualität aufweisen und eine passende Grösse vorzeigen. Wichtig ist, dass die Bilder unterschiedliche Farben aufweisen, damit man ihre Einzigartigkeit merken kann. Die Anzahl entspricht bis zu einem Spiel von 10x10 Felder und min. 5x5. Diese Bilder werden in ein Paket «Bilder» gespeichert.*

*(Änderung: Der Kunde wollte doch keine Autos oder Bilder von Tieren. Er wechselte seinen Wunsch auf Anime Kartoonbilder.)*

***3.Ziel***

*Zum guten Layout sollte auch noch ein «Beenden» Knopf platziert werden. Der Sollte, da die Benutzer dieses Spiels Kinder sein werden, gross hervorstechen und gut platziert sein. Er sollte es ermöglichen das Programm schnell zu beenden und die wichtige Spielstatistik auszugeben. Diese Wird wiederum Entweder als ein Separates Fenster ausgegeben oder im Hauptfenster. Mit dem Knopf «Beenden», soll dieses Schlussfenster auch geschlossen werden. Dies wird während der Implementation bis 24.6.21 erledigt werden.*

***Kann-Zielen***

*Das Programm mit weiteren Funktionalitäten gemäss Projektvorgabe zu vervollständigen, welches der Kunde im Optional fall wünsche. Wie zum Beispiel einen Spieltimer, Jokerkarten und Datenspeicherung zu ermöglichen. Es wäre auch noch vorteilhaft, wenn man einen Benutzernamen eingeben kann mittels GUI, anstatt nur Spieler 1/2 anzuzeigen. Wenn möglich, nach den oben erreichten Zielen erledigt, bis am 24.6.21.*

# Anwendungsentwurf

## Anwendungsfalldiagramm

## Anwendungsfallspezifikationen

|  |  |
| --- | --- |
| ID | - |
| Name | **Spielgrösse auswählen** |
| Kurzbeschreibung | Nach dem Starten mit dem «Start» Knopf, kann die Spielgrösse ausgewählt werden. Zur Auswahl 5x5 bis 10x10, diese Grösse können frei per Knopfdruck oder Eingabe bestimmt werden. |
| Akteur(e) | Spieler, er kann die gewünschte Spielgrösse auswählen |
| Auslöser | Auf die gewünschte Spielgrösse klicken |
| Vorbedingung(en) | Den «Start» Knopf gedrückt haben |
| Ergebnis | Das Memory wird gemäss Grösse aufbereitet und ausgegeben |
| Nachbedingung(en) | Das Spiel Memory fängt an. |
| Ablauf | 1. Applikation starten 2. Auf Programmbutton «Start» klicken 3. Spielgrösse auswählen (Knopf, Eingabe) |
| Alternativen | - |
| Fehlerfälle | * Die Grössenknopfe sind nicht vorhanden * De Knöpfe lassen falsche Anzahl Karten auftauchen * Spiel starten nicht oder öffnet falsche Seite. |
| Testfälle | * Die Knöpfe lassen die Anzahl gewünschten Felder erscheinen * Knöpfe sind vorhanden |

|  |  |
| --- | --- |
| ID | - |
| Name | **Statistik ausgeben** |
| Kurzbeschreibung | Klick auf den «Ende» Knopf werden die Punkte und Gewinner etc. angezeigt |
| Akteur(e) | Admin, er hat dies programmiert und so eingestellt, dass dies auf Anfrage angezeigt wird.  Spieler, er kann dies auf Knopfdruck (vom Programm) verlangen. |
| Auslöser | Druck auf «Ende» Button |
| Vorbedingung(en) | Die Spieler wollen das Spiel beenden (nach fertigem Memoryspiel) |
| Ergebnis | Ein Fenster wird angezeigt mit den Punkten |
| Nachbedingung(en) | Mit «OK» Fenster schliessen (Programm beendet) |
| Ablauf | 1. Programm starten (Button) 2. Spielgrösse auswählen 3. Spiel Spielen und Beenden 4. Ende Button drücken 5. Werte werden angezeigt (evtl. in neuem GUI) |
| Alternativen | - |
| Fehlerfälle | * Falsche Anzahl Punkte && Daten fehlen * Fenster erscheint nicht * Neue Fenster wird nicht geschlossen * Programm wird nicht beenden |
| Testfälle | * Werte und Gewinner korrekt angegeben * Fenster erscheint beim Knopfdruck |

|  |  |
| --- | --- |
| ID | - |
| Name | **Memory spielen** |
| Kurzbeschreibung | Man kann jetzt Memory spielen. Es gibt Verdeckte Karte, die man Abwechslungsweise aufdecken kann. |
| Akteur(e) | Spieler |
| Auslöser | Nachdem man die Spielgrösse ausgewählt hat, |
| Vorbedingung(en) | Spielgrösse auswählen |
| Ergebnis | Ein Fenster mit vielen verdeckten Karten wird angezeigt, mit den Namen der Spieler und den Punkten(oben). |
| Nachbedingung(en) | Danach kann entweder neu gespielt werden oder die Statistik ausgegeben werden bzw. Programm beenden |
| Ablauf | 1. Programm starten (Button) 2. Spielgrösse auswählen 3. Memory wird angezeigt |
| Alternativen | - |
| Fehlerfälle | * Keine Karten * Keine Bilder auf den Karten * Die Punkte und Namen werden nicht angezeigt * Karten lassen sich nicht umdrehen * Richtige Kartenpaare verschwinden nicht * Karten werden nach falschem Zug nicht verdeckt * Der Fasche Spieler ist an der Reihe * Wenn es Keine Karten mehr hat, wird es nicht beendet |
| Testfälle | * Karten mit Bildern vorhanden * Richtige Paare verschwinden && Anzahl neu prüfen * Falsche Paare bleiben && Anzahl neu prüfen * Die Spielerpunkte werden mitgezählt und sind korrekt * Der Spieler wird gewechselt * Bei Keinen Karten wird das Spiel beendet |

## Anforderungen

***funktionale Anforderungen***

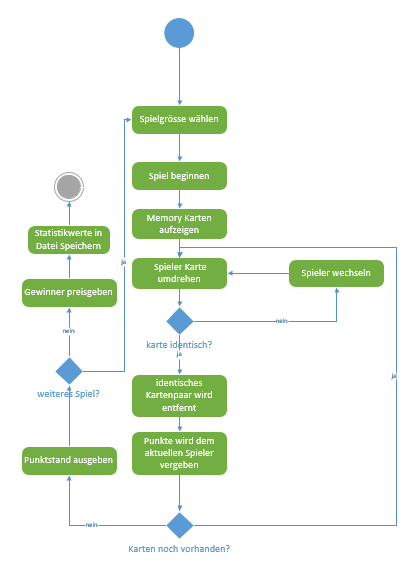
* Das Layout sollte kinderfreundlich und bunt sein
* Auf den Karten sollen Autos oder Tiere abgebildet werden(Anime)
* Die Memorykarten sind parallel nebeneinander
* Man kann die Memorykarten umkehren
* Die richtig gezogene Memorykarten sollten verschwinden
* Bei richtig gezogenen Memorykarten bleibt der aktuelle Spieler an der Reihe
* Bei Falsch gezogenen Memorykarten soll das Programm von selbst diese umdrehen
* Bei Falsch gezogenen Memorykarten muss der Spieler gewechselt werden
* Ein Knopf ist vorhanden mit dem man das Programm verlassen kann
* Das Memory muss die Punkte der Beiden Spieler zählen und ausgeben
* Der Spielstand sollte immer angezeigt werden
* Dier Ganze Spielstatistik sollte am Schluss angezeigt werden
* Das Spiel sollte mehrmals ausführbar sein
* Das Spiel kann zu jedem beliebigen Zeitpunkt beendet werden
* Spielgrösse zwischen 5x5 oder 10x10
* Optional: Zusätzliche gewünschten Funktionalitäten (Timer, Benutzername Eingabe) siehe Liste!

***Nicht-funktionale Anforderungen***

***Qualitätsanforderungen***

* Optional: Ein Zug pro Spieler dauert höchstens 15s (Timer), danach wird die Rolle gewechselt.

*Auswertung: Ich bin überzeugt, dass alle Anforderungen zu schaffen sind und ich mich darum kümmern kann. Ich denke auch, dass die Optionalen Punkte noch in der Zeit liegen werden.*

Aktivitätsdiagramm

## CRC-Karten

|  |  |
| --- | --- |
| Class: Karte  Klasse für die Karten des Memoryspiels | |
| Responsibilities:   * Liefert Art der Karten (Jokerkarten, Normal) bzw. die Punkte pro Karte. * Setzt Bilder auf die Karten (nicht Bilder laden) und liefert diese auch für das Matching der Paare. | Collaborators:  Memoryspiel |

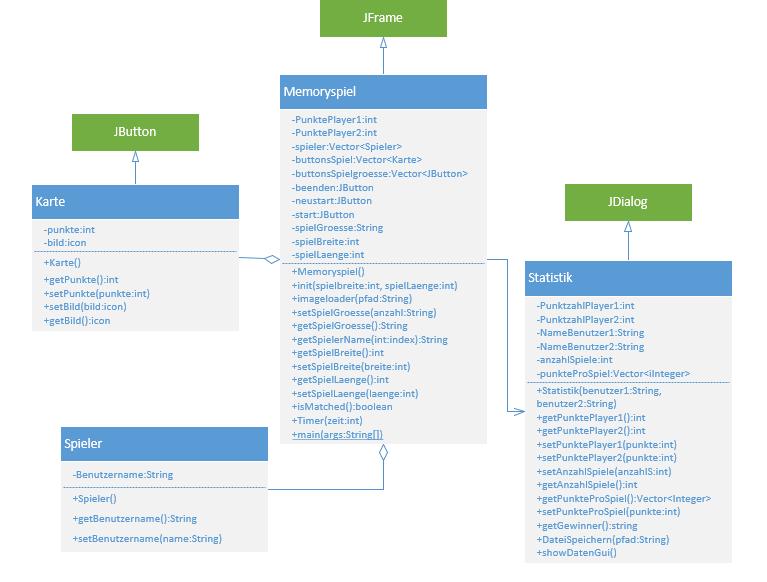
|  |  |
| --- | --- |
| Class: Spieler  Benutzer des Memoryspiels | |
| Responsibilities:   * Liefert den Benutzernamen der Spieler. | Collaborators:  Memoryspiel |

|  |  |
| --- | --- |
| Class: Memoryspiel  Geschäftslogik des Memoryspiels | |
| Responsibilities:   * Liefert Punkte pro Spieler an die Statistik. * Zählt die Punkte pro Spiel zusammen. * bestimmt die Spielgrösse/Fläche des Spiels. * Erstellt Knöpfe mit den Bildern. * Ladet die Bilder und liefert diese an die Klasse «Karte». * Zuständig für Neustart und beenden des Programms. * Regelt Spielregeln: limitierte Zeit für einen Zug, falls richtige Kartenpaare gezogen ist der gleiche Spieler am Zug etc. * Lässt den Benutzer mit dem Spiellogo begrüssen. | Collaborators:  Karte, Statistik, Spieler |

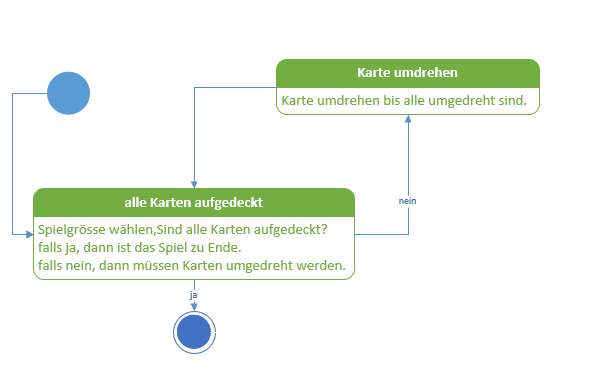
|  |  |
| --- | --- |
| Class: Statistik  Klasse für das Anzeigen und in einer Datei zu speichernde Statistik. | |
| Responsibilities:   * Verwaltet die von Memoryspiel übertragenen Spieldaten und zählt diese zusammen (Zusammenzählen könnte man auch in der Klasse Memoryspiel, da wir aber noch nicht wissen was besser ist, werden wir dies testen und dann anpassen). * Vergleicht die Punktzahlen der Spieler und liefert dann einen Gewinner (mit dem Benutzernamen) und die gesammelten Punkte beider Spieler. * Speichert alle Spieldaten des Memorys in eine Datei. Im Gui wird der Pfad (Speicherort) angezeigt, * Liefert die Anzahl gespielter Spiele. | Collaborators:  Memoryspiel |

## Klassendiagramm

*Siehe nächste Seite 9 für Erklärung!*



* Vorwort: Ein richtiges Klassendiagramm zu Zeichen ist sehr schwierig, da man Entweder viele Möglichkeiten hat etwas umzusetzen, sodass man gar nicht weiss welche Methode man nehmen soll oder man hat gar keine Ahnung wie man es umsetzten kann.
* Unser Klassendiagramm: Rechts befindet sich die Klasse Karte, welche dann unsere Memorykarten sein werden. Spieler Klasse soll genutzt werden, um die zwei Spieler zu erstellen und ihnen einen erkennbaren Benutzername zu geben. Die Klasse Statistik wird genutzt, um die Daten auszuwerten. Wir dachten an ein JDialog der auf Befehl der klasse Memoryspiel aufgerufen wird. In der Mitte befindet sich die wichtigste Klasse nämlich das Memoryspiel. Dort drin werden Methoden ausgeführt wie ob die Karten einen Match haben, die Spielgrösse wird gesetzt, Bilder werden geladen und ein Timer ausgeführt. Bei dieser Klasse sind wir aber über Korrektheit noch nicht sicher, vor allem das mit den Bildern laden und Punkten übergeben. Dies wird sich hoffentlich im Laufe noch ergeben.

Zustandsdiagramm

Beim MemorySpiel gibt es genau zwei Zustände. Alle Karten sind aufdeckt oder man Deckt diese zu, sodass sie den Karten Umdrehen Zustand einnehmen. Ein Zug findet so statt: Der Spieler1 dreht eine Karte auf, falls die Kartenpaare gleich sind, werden diese entfernt und der Spieler1 kann weiter Karten umdrehen, falls es noch hat. Wenn er eine Falsche Karte gezogen hat, wird der Spieler gewechselt, dieser kann auch nach gleichem Schema Karten ziehen.

## Sequenzdiagramm

## Ganz Rechts werden zwei Spieler mit Verschiedenen Benutzernamen erstellt. Wichtig ist anzumerken, dass alle Methoden ausgehend der Klasse Memoryspiel in der «Main» Methode ausgeführt bzw. aufgerufen werden. Die Memoryklasse Kommuniziert mit der Klasse Karte, wobei diese (Memoryspiel) die Bilder ladet und sie in der Methode init() mit der setBild() Funktion auf die Karten setzt. Danach mit der Spielgrösse das Spiel erstellt und noch den Zugtimer(Spielerzug) gesetzt. Das Spiel beginnt. Mit der isMatched() Methode schaut das Programm ob der Benutzer richtige Kartenpaare gezogen hat. Er Kontrolliert dies mit der getBild() Methode und Je nach Richtigkeit werden mit der getPunkte() Methode die richtige Anzahl Punkte verteilt(0,10,Joker=20). Zum Schluss wird die Statistik angezeigt und in einer Datei gespeichert.

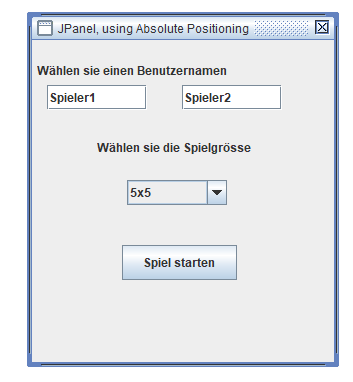
## Mockups

**Willkommensfenster**

# 

1. Dies ist die Begrüssung bzw. Willkommens GUI des Memoryspiel. Dem Benutzer wird beim starten des Programms das schöne Logo des Memoryspiels angezeigt und kann zum Starten des Spiels den «Get Started» Knopf betätigen, welcher sich unterhalb befindet. Daraufhin wird man zum 2. Fenster weitergeleitet. Hier wird für die Platzierung ein Borderlayout genutzt welches mit JPanels für den Abstand und Zentrierung der Komponenten zuständig ist. Unsere Komponenten werden hierbei JLabel(mit Imageicon) und JButton sein.

**Einstellungsfenster**

1. Als nächstes gelangt man zum Einstellungsfenster, bei dem man wichtige Einstellungen treffen kann. Oben kann man hier entsprechende Namen für die Spieler geben. Beispielsweise «Tenzin124» etc. Falls man keinen eigenen Namen benutzen möchte, werden, wie im JTextfield gezeigt, die vorgeschlagen Namen übernommen. Spieler1 und Spieler2. Unten kann man dann noch seine gewünschte Spielgrösse mittels JCombobox auswählen. Die mögliche Auswahl liegt hierbei zwischen 5X5 bis 10x10. Wenn der Benutzer sich für einen Namen und eine Grösse entschieden hat, kann dieser mit dem Knopf «Spiel Starten» das Memoryspiel beginnen. Hier werden wir auch Borderlayout mit JPanels für die Umsetzung benutzen. Bei den JTexfield könnten wir bei Bedarf noch ein Gridlayout benutzen. Für die Texte werden JLabels verwendet.

**Memoryspie**l

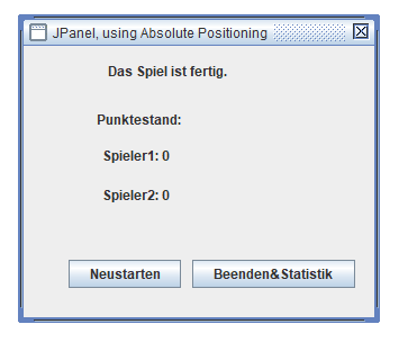
# 

1. Das Main-GUI. Jetzt hat das Memoryspiel begonnen und die Spieler 1 und 2 können Karten umdrehen. In der Mitte kann man diese sehen, nämlich 25 Memorykarten. Diese sind von der vorherigen Einstellung (Spielgrösse 5x5) realisiert worden. Die Karten sind JButtons, welche umgedreht werden, wenn man auf sie klickt. Falls die Kartenpaare übereinstimmen verschwinden(ausgeblenden) diese. Oben sieht man die Namen der Spieler und deren Punktstand, anfangs auf 0 gesetzt. Der Name des Spielers, welche mit einer roten Umrandung gekennzeichnet ist, ist gerade am Zug (er darf Karten umdrehen). Das heisst, der Spieler, der an der Reihe ist, muss klar ersichtlich sein. Dies wird entweder so im Programm umgesetzt, dass die Farbe des Benutzernamen (JLabel) verändert oder den Hintergrund hinter dem Namen hervorstechen wird.

Oben in der Mitte ist ein Timer, welche die verbleibende Zeit für ein Zug, pro Spieler anzeigt (welcher am Zug ist). Unten rechts hat es ein JButton «Beenden». Mit diesem kann man das Spiel vorzeitig beenden, falls man nicht weiterspielen möchte. Wenn alle Karten umgedreht wurden, wird das GUI automatisch auf das 4. GUI springen, nämlich zum Zwischenstand.

Für die Karten benutzen wird ein Gridlayout. Für die Punkteleiste oben und unten werden Borderlayouts mit JPanels genutzt. Für den Text JLabels.

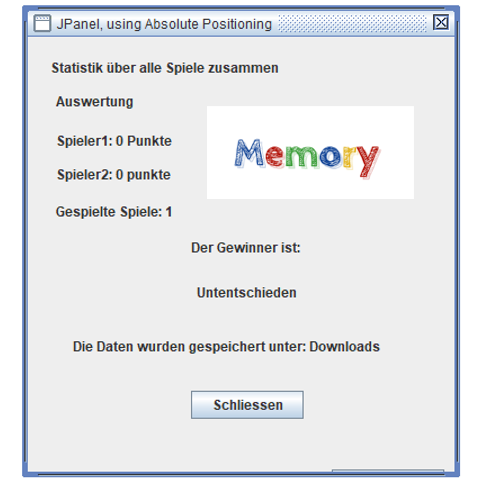
**Zwischenstand**



1. Immer wenn man ein Memoryspiel beendet oder fertig gespielt hat, wir ein kleines Zwischenfenster angezeigt nämlich das Zwischenstandfenster. Dieses zeigt die erreichte Punktzahl im aktuellen Memoryspiel an. Spieler 1 und 2 können sich somit dauernd vergleichen und haben einen Anreiz nächstes Spiel zu besser zu sein.

Ab Hier gibt es zwei Möglichkeiten entweder man startet ein Spiel neu mit dem «Neustarten» JButton (zurück zum 2. Fenster) oder man will nicht mehr Memory spielen und beendet das Programm ganz mit dem «Beenden&Statistik» JButton (der einen zum 5. Fenster: Schlussstatistik sendet). Für die Umsetzung wird hier ein Borderlayout mit JPanels genutzt. Für die Knöpfe ein Gridlayout. Für den Text JLabels.

**Schlussstatistik**



1. Nachdem man sich Entschieden hat das Memory ganz zu beenden, wird zuerst die Schlussstatistik ausgegeben. Sie zeigt Daten an wie: Wie viel punkte ein Spieler insgesamt erreicht hat, wie viel spiele insgesamt gespielt wurden und wer von beiden mehr Punkte hat und somit zum Sieger geehrt wird. Im Hintergrund werden zudem, neben diesen oben erwähnten Daten, noch zusätzlich die Punkte pro Spiel gespeichert und diese in eine Externe Datei geschrieben (mit den Benutzernamen etc.). Diese ist, wie im GUI angeben, unter diesen bestimmten Pfad zu finden. Beim Druck auf den JButton «Schliessen» wir das Programm ganz beendet und das Fenster geschlossen.

Für die Umsetzung wird hier ein Borderlayout mit JPanels genutzt. Für den Text und Bilder JLabels. (Edit: Es wurde ein kleineres Gui realisiert)

# Testfälle

## JUnit-Testfälle

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-001 |
| Kurzbeschreibung | Testet ob der Defaultname(Spieler1/2) benutzt wird wenn der Benutzername nicht geändert wird. |
| Testdaten | Der Spielername, der nicht vom Spieler geändert wurde. |
| Ablauf | Die Spieler lassen das Kästchen mit dem Benutzernamen sein und ändern nichts. Sie Starten das Memoryspiel. |
| Erwartetes Testresultat | * Der Name der Spieler wird als Spieler1 und 2 gespeichert. * Es sind zwei Namen vorhanden |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-002 |
| Kurzbeschreibung | Testet ob bei Leerer Eingabe des Benutzernamens, der Defaultname(Spieler1/2) übernommen wird. |
| Testdaten | Vorgeschlagener Default Name für den Benutzernamen wird durch Leerer String ersetzt (Text in der JTextfield gelöscht) |
| Ablauf | Die Spieler ändern den Namen zu leer(nichts) und starten das Spiel. |
| Erwartetes Testresultat | * Der Name der Spieler wird als Spieler1 und 2 gespeichert. * Es sind zwei Namen vorhanden |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-003 |
| Kurzbeschreibung | Testet ob bei Eingabe des eigenen Benutzernamens, dieser übernommen und auch gespeichert wird. |
| Testdaten | Die Spieler geben einen Eigenen Namen ein |
| Ablauf | Die Spieler ändern den Namen im EinstellungsGUI und starten das Spiel. |
| Erwartetes Testresultat | * Die eigebenden Namen für die Benutzern werden gespeichert. * Es sind zwei Namen vorhanden |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-004 |
| Kurzbeschreibung | Testet ob genauso viele Karten erzeugt werden wie ausgewählt. |
| Testdaten | Die Memorykarten, die aufgrund der Spielgrösse erstellt wurden. |
| Ablauf | Der Spieler wählt eine Spielgrösse und startet das Memoryspiel. |
| Erwartetes Testresultat | Es wurden so viele knöpfe erzeugt, wie Spielgrösse ausgewählt. |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-005 |
| Kurzbeschreibung | Testet ob die richtigen Spielgrössen vorhanden sind. |
| Testdaten | Spielgrössen in einem Vector gespeichert |
| Ablauf | Der Spieler will die Spielgrösse auswählen, dafür hat er eine JCombobox |
| Erwartetes Testresultat | * Es sind 5 Werte im Array vorhanden * Die werte sind 5x5 bis 10x10 (Gleiche Zahlen links und rechts neben x) |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-006 |
| Kurzbeschreibung | Es gibt Karten die mehr Punkte geben |
| Testdaten | Joker karten, die mehr Punkte geben. |
| Ablauf | Karten werden erstellt(intern) |
| Erwartetes Testresultat | Es gibt Karten die mehr Punkte geben ->Jokerkarten |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-007 |
| Kurzbeschreibung | Es gibt genau so viele verschiedene Bilder, dass es genug hat für ein Memoryspiel (Bilder mal 2 =gesamt, alle Karten) |
| Testdaten | Die Pfade der Bilder in einem Vector |
| Ablauf | Die Bilder werden anhand der Pfade beim Aufbereiten des Spiels in einem Vector gesetzt. (imageloader()) |
| Erwartetes Testresultat | * Es gibt genau so viel pfade wie benötigt. * (Pfade mal 2) = gesamte Anzahl Karten |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-008 |
| Kurzbeschreibung | Unzulässiger Benutzername (z. B: links, ungültige zeichen), der vom Spieler eingeben wird. |
| Testdaten | Benutzername der nicht zulässig ist |
| Ablauf | Der Spieler verwendet unzulässige Zeichen als Benutzernamen. |
| Erwartetes Testresultat | Es sollte eine Fehlermeldung erscheinen und das Spiel sollte nicht gestartet werden. |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-009 |
| Kurzbeschreibung | Karten werden gemischt |
| Testdaten | Die Memorykarten |
| Ablauf | Der Spieler erstellt ein neues Spiel und die Karten sind immer unterschiedlich angeordnet |
| Erwartetes Testresultat | * Die Karten weisen immer unterschiedliche Bilder auf. * Pfade von den Bildern auf Knöpfe ändern sich immer wieder. |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-010 |
| Kurzbeschreibung | Timer ist auf 15 Sekunden beschränkt |
| Testdaten | Der Timer mit den eingestellten 15 Sekunden |
| Ablauf | Beim Starten des Memoryspiels erscheint ein Timer, der runter zählt(pro Zug des Spielers). |
| Erwartetes Testresultat | Timer fängt bei 15 Sekunden an (wert =15). |

*(Bemerkung: Testfall TF-010 wurde von 20 sek auf 15 sek reduziert, das 20 zu viel sind)*

## Manuelle Testfälle

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-011 |
| Kurzbeschreibung | Testet, ob das Design des GUI gleich wie das Mockup ist. Bild und Knopf in der Mitte, mit Abstand zueinander und den Seiten. |
| Testdaten | WilkommensGUI mit den Komponenten |
| Ablauf | Programm wird gestartet |
| Erwartetes Testresultat | * Bild ist in der Mitte des GUIs * Das Bild ist scharf und hat eine angemessene Grösse * Der Knopf ist beschriftet und unter dem Bild platziert * Die Komponenten haben Abstand zueinander. |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-012 |
| Kurzbeschreibung | Testet, ob der Spieler, der an der Reihe ist, im GUI markiert wird. |
| Testdaten | Die Memorykarten |
| Ablauf | Memory wurde gestartet. Der Spieler ist an der Reihe |
| Erwartetes Testresultat | * Das GUI zeigt mit farbigen Markierungen wer an der Reihe ist. * Es wird korrekt gewechselt (Beachtung von Regeln wie der Timer9) |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-013 |
| Kurzbeschreibung | Testet, ob bei ungerader ausgewählte Spielgrösse, gerade Anzahl Karten freigeschaltet werden. |
| Testdaten | Memorykarten die angezeigt werden |
| Ablauf | Die ungerade Spielgrösse ist ausgewählt worden und das Memoryspiel wurde gestartet. |
| Erwartetes Testresultat | * Es werden gerade Anzahl Memorykarten angezeigt bei ungerader Spielgrösse * Die Memorykarte, welche zu viel ist, wird deaktiviert. |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-014 |
| Kurzbeschreibung | Testet, ob nach dem Aufdecken aller Karten, automatisch das ZwischenstandGUI geöffnet wird. |
| Testdaten | Memorykarten die alle aufgedeckt wurden |
| Ablauf | Alle Karten des Memoryspiels wurden aufgedeckt durch die Spieler. |
| Erwartetes Testresultat | * Es erscheint ein neues Fenster mit den Statistiken des vorherigen Spiels * Die Ergebnisse werden korrekt angezeigt. (Punktzahlen, Benutzernamen) * Alle Komponenten werden angezeigt (JButtons/JLabels) |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-015 |
| Kurzbeschreibung | Beim Neustarten des Programms, gelangt der Benutzer vom ZwischenstandGUI zum EinstellungsGUI |
| Testdaten | Neustartbutton |
| Ablauf | Der Spieler drückt den Neustartbutton |
| Erwartetes Testresultat | * Es erscheint das EinstellungsGUI. * Man kann wieder auswählen welche Grösse man will * Die Benutzernamen sind vom letzten Eintrag noch gespeichert (falls selber benannt). |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-016 |
| Kurzbeschreibung | Es wird getestet ob bei erneuten Spielstart, die korrekte Anzahl Karten ausgeben (GUI) wird und die alte Anzahl an Knöpfe vom GUI entfernt wurde. |
| Testdaten | * MemoryspielGUI * Andere Spielgrösse auswählen |
| Ablauf | Der Spieler startet das Spiel neu, wählt eine andere Spielgrösse als vorher und startet das Memory. |
| Erwartetes Testresultat | * Im MemoryGUI sollte die richtige Anzahl Knöpfe Karten erscheinen * Die Alten Knöpfe wurden entfernt bzw. ausgeblendet. |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-017 |
| Kurzbeschreibung | Karten Bilder werden angezeigt |
| Testdaten | Die Memorykarten mit den Bildern |
| Ablauf | Der Spieler startet das Memory |
| Erwartetes Testresultat | * Es erscheinen verdeckte Karten mit verschiedenen Bildern. * Die Bilder nehmen die Grösse der JButtons (Karten) an. |

# 

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-018 |
| Kurzbeschreibung | Wenn der Spieler ein richtiges Kartenpaar gefunden hat, sollten diese Karten automatisch ausgeblendet werden. |
| Testdaten | Die Memorykarten mit den Bildern |
| Ablauf | Der Spieler deckt zwei gleich aussehende Kartenpaare auf. |
| Erwartetes Testresultat | * Die Karten werden ausgeblendet * Der jeweilige Spieler erhält punkte und ist noch an der Reihe. * Der Timer hat von vorne begonnen |

|  |  |
| --- | --- |
| Element | Beschreibung (Beispiele) |
| Testfallnummer | TF-019 |
| Kurzbeschreibung | Wenn der Spieler ein falsches Kartenpaar gefunden hat, sollten diese Karten automatisch wieder umgedreht werden. |
| Testdaten | Die Memorykarten mit den Bildern |
| Ablauf | Der Spieler deckt zwei nicht gleich aussehende Kartenpaare auf. |
| Erwartetes Testresultat | * Die Karten wurden zurückgedreht und nicht ausgeblendet * Der andere Spieler ist an der Reihe. |

# Lösungsbeschreibung

Unser Programm wurde mittels der oben erwähnten Klassen im Klassendiagramm umgesetzt. Dieses war zwar nicht perfekt umgesetzt worden, doch reichte, um sich daran zu orientieren. Es stellte also ein Gutes Grundgerüst dar. Auch das Sequenzdiagramm ging zwar in die richtige Richtung, aber die Reihenfolge der Methodenaufrufe war ein bisschen falsch.

**Grund-Ziele**

Die Grundziele wie z.B, dass man eine Memorykartengrösse auswählen zwischen 5x5 und 10x10 kann oder dass sich die Karten umdrehen, falls der Spieler falsche Paare erwischt hat, wurden alle umgesetzt.

Ein Passendes Layout wurde auch gesetzt, hätte aber hätte auch noch schöner sein können und Kinderfreundlicher. Aber für diese Anwendung völlig ausreichend.

**Soll-Ziele**

Wir haben es sogar geschafft, alle zusätzlichen Funktionalitäten gemäss Soll-Ziele umzusetzen und dies noch der geplanten Zeit. Diese beinhalten: Es gibt zu jedem Spiel, egal wie gross, genau 3 verschiedene Joker Karten, die mehr Punkte geben. Dazu haben wir die Punkteverteilung so gesetzt: 40 Punkte für Korrekt gefundene Joker kartenpaare und 20 für korrekt gefundene Normale Kartenpaare. Somit wurde das Ziel mit den Jokerkarten erfüllt.

Auch gibt es einen Timer, welcher mittels Aktualisierbares JLabel umgesetzt. Z.B 15 Sekunden werden auf dem GUI angezeigt, durch das repaint() wird diese Zahl runter gezählt, sodass es aussieht wie ein Timer. Nachdem diese 15 Sekunden abgelaufen sind, wechselt es Automatisch den Spieler und der Timer fängt von vorne an bzw. wenn der Spieler eine einen richtigen Zug gemacht hat, bleibt er noch bei diesem Spieler.

Auch die Datenspeicherung wurde umgesetzt und funktioniert tadellos. Es wird eine Leeres File erstellt und danach alle Ergebnisse für die Auswertung in diese Datei geschrieben. Z.B Benutzernamen, Datum, Punkte pro Spiel pro Spieler, Gesamtpunktzahl der Spieler und der Gewinner.

**Anforderungen**

Es fast alle Anforderungen eingehalten. Nur das Layout hätte noch bunter sein können und wir haben uns am Schluss für eins andere Bilderthema der Memorykarten entschieden. Anime anstatt Autos oder Tiere. (Mehr dazu bei Änderungen)

**Änderungen**

Wir planten als Memorykarten Bilder Autos oder Tieren zu erstellen. Der Kunde wünschte sich das als Ziel bzw. Anforderung. Wir haben durch das Einverständnis des Kunden zu «Anime» Bilder gewechselt, weil diese aufregender sind als Tiere oder Autos. Diese Karten besitzen einen grösseren Kontrast und sind einfacher zu merken.

Das StatistikGUI wurde auch anders umgesetzt. Es wurde kleiner gehalten, weil die Daten sehr unübersichtlich waren und wir uns deshalb entschieden haben auf gewisse Datenwerte auf dem StatistikGUI zu verzichten. Sie werden deshalb nur in dem File Auswertungen.txt geschrieben.

Das mit dem Zusammenzählen der Punkte pro Spiel in der Klasse Memorykarten wurde in der Klasse StatistikGui umgesetzt, weil es logischer und übersichtlicher ist.

Der Timer wurde beim Testfall TF-010 von 20 auf 15 Sekunden reduziert, da 20 zu viel sind.

**SOLL-IST Vergleich**

|  |  |
| --- | --- |
| Soll | IST |
| 5x5 bis 10x10 Spielgrösse | Vorhanden |
| Beenden Button | Vorhanden |
| Timer 20 Sekunden | Timer 15 Sekunden |
| Benutzername Eingabe | vorhanden |
| Memorybilder und Logo | Vorhanden |
| Gemäss Mockup realisiert | Ja, ausser StatistikGUI |
| Bilder von Tieren und Autos | Bilder vorhanden, aber von Anime |

Ich könnte jetzt noch weitere vergleiche aufzählen, aber die Liste wäre zu lang. Damit will ich sagen, dass ausser bei den Änderungen oben, alles erfüllt wurde und so umgesetzt, wie es definiert wurde vom SOLL zu IST Zustand. Es wurde auch mehrfach getestet und es entspricht 1:1 zum erwarteten Ergebnis.

**MVC**

Wir haben wenig mit Entwurfsmuster gearbeitet bzw uns daran orientiert. Aber man kann sagen das die Memorykartenklasse die View darstellt und Controller beinhaltet. (Auch wenig Data beinhaltet). Aber die richtige Data ist in diesem Fall unser StatistikGUI welches die Daten für die Auswertung durch die View bzw. durch den Controller des Memoryspiels bekommt und dieses in einem Vector speichert. Dies beinhaltet somit alle wichtigen Daten.

**Weiterentwicklungsmöglichkeiten**

Weiterentwicklungsmöglichkeiten für das Programm wären, dass es mehr Anpassungsmöglichkeiten bezüglich des GUI gibt, die der Benutzer auswählen kann. Z.B Kartengrösse wechseln, Kartenbreite, Kartenlänge, Farben des GUI ändern oder eigene Bilder mittels Internetlink abrufen. Es sollte auch noch ein Spielbot erstellt werden gegen den ein Einzelspieler Spielen kann. Es sollten auch weitere Spielgrössen zur Auswahl hinzugefügt werden. Beispiel 4x5,6x8 etc.

# 

# Anhang

* Testprotokolle
* API (im Memoryspiel/Src/API Ordner)
* Benutzerhandbuch
* Memorybilder (im Memoryspiel/src/Bild Ordner)
* Memory Programm (im Memoryspiel/src)
* Testfälle JUnit (in Memoryspiel/Src)
* Auswertungen Memoryspiel (Memoryspiel/src/Auswertung)
* Deklaration Eigenleistungen