Dokumentation „MemoryGame“

von der Gruppe „Proggen“

Inhaltsverzeichnis

Einführung

In den folgenden Zeilen beschreiben wir Ihnen, wie das Spiel „Memory“ aufgebaut wird und wie es funktioniert.
Das Spiel ist ein eins bis zwei Spieler Programm, welches eine grafische Oberfläche besitzt und 20 Felder beinhaltet.
Man kann es entweder alleine oder mit einem weiteren Mitspieler spielen. Das Spiel wird aufgebaut durch die Visualisierung, die Mechanik, die GUI und den SeverClient. Die „Spielregeln“ sind ganz klar; Gewinner ist derjenige, der die meisten Pärchen aufgedeckt hat.
Das MemoryGame kann man dank dem ServerClient auf zwei unterschiedliche Monitore spielen, welches vieles erleichtert hat.

Mechanik

Es wurde eine Klasse erstellt, die den Namen „Memoryfield“ enthält.
In dieser Klasse wurde, mit Hilfe des „private final int...“, 2*10 Zeichen erstellt.
Zu der Klasse wurde ein boolean erstellt, wobei ein „new boolean“ dazu führte, dass das boolean eine gewisse Menge an Speicher erzeugt bekommen hat, damit die Visualisirung eine Spaltenlänge von 4 und eine Zeilenlänge von 5 Arrays erzeugen kann. In der verketteten Liste von String (StringBuffer) werden unterschiedliche kleine Buchstaben durch Hilfe von ((char)(int)...) in dem Stringbuffer eingespeichert.
Der „randomString“ erstellt die jeweiligen Buchstaben.
Der „Math.random“ ist dafür da, damit es Zufallswerte raus geben kann und „*26)+97));“ ist die jeweilige Schreibweise, damit man sich auf klein-Buchstaben beziehen kann. Wenn man Beispielweise die „97“ in diesem Algorithmus verändern würde, wird eine andere Zeichenkette ausgegeben.
In nächsten Abschnitt sorgt die Methode „randomString2“ dafür, dass die Chars in der ersten Methode „randomString“ nicht doppelt vorkommen. Sollten weiterhin doppelte Chars bestehen, wird somit in der Methode ein weiteres String erstellt, der dies behebt.
Anschließend wird eine Klasse „Matchfield“ initialisiert, welches dafür sorgt, dass die Buchstaben in den jeweiligen Feldern eingespeichert werden.
In den nächsten Vorgängen werden zwei Integer implementiert.
„rand1“ wird erstellt und das Randomwert einen Wert von höchstens 0-3 annhemen kann.
„rand2“ wird erstellt und das Randomwert einen Wert von höchstens 0-4 annhemen kann.
Der Index wird zu Anfang auf 0 gesetzt, welches für den Lauf Index für randomString gedacht ist.
Es wird eine While-Schleife implementiert. In dieser Schleife werden, solange alle Werte nicht belegt sind, folgendes durchgeführt; und zwar wird geprüft, ob an der Stelle „rand1“ und „rand2“ von „boolArray“ kein Wert belegt ist. Sollte dies belegt sein, wird ein Char an der Stelle des Index von „randonString“ ins Feld gespeichert.
An der gleichen Stelle wie „rand1“ und „rand2“ wird diese Belegung auf „True“ gesetzt.

Anschließend werden neue Werte gesetzt. Falls das Array die Grenze überschritten hat, wird der Index angepasst und soll wieder auf 0 gesetzt werden und von vorne beginnen, andernfalls soll der Index sich um einen erhöhen.

Die Klasse „Player“ wird erstellt. In dieser Klasse werden der jeweilige Spieler erzeugt. Zusätzlich wird sein Name festgehalten und auch, ob er am Zug ist. Bzw. ob er an der Reihe ist zu spielen.
Der Name wird anschließend dem Spieler zugewiesen. Durch den Boolean wird entscheiden, wer momentan an der Reihe ist und dazu werden die derzeit erhaltenen Punkte gesetzt.

GUI

Nun kommen wir zu der Klasse „GUI“. Sie dient hauptsächlich für die Visualisierung des Feldes. Die jeweiligen Label werden für die Button & Co. Initialisiert. Außerdem werden die Werte für einzelne Methoden in der GUI implementiert.
Die GUI übernimmt, abgesehen von der grafischen Darstellung, auch wesentliche Bestandteile der Spiellogik. Als nächstes wird die MemoryField-Klasse neu erzeugt.
Hierbei wird ein CharArray, mit einer unterschiedlichen Reihenfolge von Zeichen, erzeugt.
Anschließend wird ein neues CharArray erzeugt, der die gleiche Größe besitzt, wie im MemoryField. Dadurch, dass ein SpielFeld erzeugt wurde, werden diese in die mainCards Variable eingespeichert.
Für die ZweiSpieler Funktion wurde die Methode SetMainCards und GetMainCards angewendet. Die SetMainCards-Methode sorgt dafür, dass die im Parameter übergebenen charArray in die GUI mainCards-Variable eingesetzt werden. Die GetMainCards-Methode gibt die MainCards Variable von der GUI durch einen „return“ aus.
Es wird in den nächsten Zeilen die „Scene“ für das Memory Feld implementiert und die einzelnen Bestandteile der Visualisierung werden auf die Scene initialisiert, zum Beispiel werden die Label, die Button und alle anderen wichtigen Dinge der Visualisierung hinzugefügt.
Sobald dies getan wurde, wird die Scene ausgegeben. Dazu werden die Label und die Button, die in der Scene nun enthalten sind, erzeugt. Zusätzlich wird die Schriftgröße und die Schriftart der Label, die Größe der einzelnen Button und die Größe der Scene durch eine X-Achse und eine Y-Achse implementiert. Dies kann man sich wie ein Koordinatensystem vorstellen. Die X-Achse wurde auf 200 gesetzt und die Y-Achse auf -200. Alles andere werden in diesem Koordinatensystem angepasst.
Die Problematik in diesem Bereich war es, dass wenn man ein Button gedrückt hat, die StackPane ein neue Scene erstellt hat, aber die alte Scene nicht geschlossen wurde. Dies wurde doch trotz allem erfolgreich umgesetzt, sodass das alte Fenster sich schließt, wenn das neue sich öffnet. Dies haben wurde jedoch in der Klasse „Main“ umgesetzt. Dazu später mehr.

für das Memory-Feld, den Memory-Menü, Zweispieler-Menü und den Join-Button musste jeweils eine Scene erstellt werden.
Als nächstes wird eine Methode erstellt, der die einzelnen Button zeigt und die Punktzahl der Spieler im Online Modus fest legt. Dies wird als „Synchronized“ festgehalten, sodass nur ein Thread diese Methode abarbeiten kann.
sollte das ButtonResetTime-Thread am laufen sein, so wird das drücken verhindert und man kann keine weiteren Buttons drücken.
Anschließend gilt diese Bedingung, wenn ein Spieler am Zug ist und entsprechend der Interface Player, das heißt, wenn die Abstrakte Klasse nicht auf null gesetzt wurde, so wird ButtonClicked von „Wert1“ und „Wert2“ zum nächsten Spieler versendet, um die gedruckten Buttons auch bei ihm anzeigen zu lassen. Sollte Beispielweise Spieler1 dran sein, so muss Spieler2 warten, bis er an der Reihe ist und darf somit keine Button drücken. Zusätzlich gilt diese Bedingung für das erste Klicken eines Button.
Sollte der Fall eintreffen, dass ein Button geklickt wurden, so wird geprüft, ob dieser Button das erste oder das zweite Button, mithilfe des „Counters“ ist. Das bedeutet das wenn der erste Button geklickt wurde, der Counter somit auf „1“ gesetzt wird, um zu zeigen, dass ein Button geklickt wurde.
Das Zeichen des MemoryField wird in das Button an der Stelle "Wert1" und "Wert2" gesetzt. Die Belegung an der Stelle wird auf „True“ gesetzt und "Wert1" und "wert2" wird zwischen gespeichert, um einen Vergleich erzielen zu können, mit dem zweiten Button-Klick. Sobald der zweite Button betätigt wurde, wird in der Methode geprüft, ob die beiden Button identisch zueinander sind oder eher weniger. Sollten sie identisch sein, so bleiben beide Button angezeigt und auf „True“ gesetzt und die Punktzahl des Spieler wird demnach ebenfalls angepasst und um eins erhöht, da derjenige einen Punkt erzielt hat.
Die Methode, updated das „TurnLabel“ für das Online Spielen. Das heißt, je nachdem welcher Spieler dran ist, so wird das TurnLabel im Spielfenster des Online-Spiel angepasst. Es gibt nur Host und Client: Host= Server-ersteller. Client= Der mit Server verbindet. Sollten alle Felder belegt sein, so wird eine Abfrage erstellt, die befragt, ob man das Spiel neugestartet werden solle oder nicht.
Dies geschieht mit Hilfe eines Dialog Fensters.

Als nächstes wird eine weiter Klasse erstellt und zwar die Klasse „ButtonResetTimer“. In dieser Klasse werden die Buttons, die nicht zueinander passen, weiterhin auf „false“ bleiben und anschließend gelöscht werden vom Feld.
Der Konstruktor erwartet als Eingabe, das erste Button, welches betätigt wurde und das Zweite. Außerdem wird eine bestimmte Zeitangabe von einer Sekunde implementiert, damit, falls zwei Button geklickt wurden, man in der Zeit keine anderen Felder aufdecken kann und man somit nicht in Versuchung kommt, zu mogeln. Sollte die Zeitangabe erreicht sein, so werden die Button wieder von der Bildfläche gelöscht und man kann nun einen weiteren Button drücken.
Um die Methode „unlockNewClick()“ aufrufen zu können, benötigt man die GUI, damit die Button während der Zeitangabe angezeigt bleiben.

Es wird eine letzte Klasse zu der GUI initialisiert und das ist die Klasse „Main“.
Diese Klasse dient für das gesamte Spiel Fenster.
In dieser Klasse wird das Memory-Menü-Fenster angezeigt. Zzuästzlich werden die jeweiligen Funktion dews Spieler1 und Spieler2 erzeugt. Außerdem wird die Scene für das Memory-Feld angepasst und erzeugt.
Der ResetButton wird hier, wenn eine Belegung festgestellt wurde, so wird mithilfe eines Dialog Fensters abgefragt, ob man zurück zum Menü möchte, sollte man dies mit einem „JA“ bestätigen, so wird das Spiel neu gesetzt. Andernfalls wird nichts gedruckt, wenn man keine Reaktion gegenüber dieser Frage zeigt. So gelangt man ohne Dialog Fenster zum Hauptmenü. Danach wird der Neustart-Button erzeugt, sodass der Spieler das Spiel von vorne spielen kann. Zusätzlich wird die Funktion für den Host-Button und den Join-Button erzeugt.

Server

Zunächst wird der Server mit der Klasse GameClients und den Parametern IP-Adress und myGUI erstellt. Danach wird die Methode server.connect implementiert. Diese hat die Aufgabe sich mit dem Server zu verbinden. Daraufhin wird ein BufferedReader und ein PrintWriter erzeugt. Sobald eine Nachricht verschickt wird, wird mit Hilfe des BufferedReader die Nachricht gelesen und abgearbeitet. Als nächstes wird ein StringTokenizer erstellt. Mit dem StringTokenizer wird die Nachricht in eine Liste von String gesetzt. Hier wird das Beispiel ,,field absdefg" genommen. Sobald der Reader erkennt, dass das ,,field" gelesen wurde, wird das ganze Feld vom Host in das Feld vom Client gesichert. Wenn der Reader Player liest, dann wird der Player vom Client mit den Vorgaben vom Host neu initialisiert. Sollte der Client dabei nicht dran sein, so wird der TurnHandler gestartet. Der TurnHandler nimmt dabei den Reader und die GUI vom jeweiligen Spieler, damit diese gelesen wurden und angezeigt werden kann. Als nächstes wird die Klasse ThreadServer implementiert, welche von der abstrakten Klasse InternetPlayer erbt. Hierbei erbt er gleichzeitig von Runnable, welches dafür sorgt ein Thread zu starten.

Im Konstruktor wird bereits der Server mit der eigenen IP-Adresse gestartet. Der Server versucht nun sich mit der eigenen IP-Adresse zu verbinden, andernfalls gibt er eine Fehlermeldung, welche ,,Exception" ist, aus. Falls sich der Server mit der lokalen IP-Adresse verbunden hat, verbindet er sich auch gleichzeitig mit der Portnummer 5000.

Die Methode versendet den Player Zug vom Host. Somit kann der Spieler beziehungsweise der Client seinen Zug anpassen. Wenn aber der Host Spieler nach dem Zufallsprinzip dran ist, dann ist er Client nicht dran.

Des weiteren wird eine Klasse ,,sendPlayer" erstellt. Dieser schreibt dem Client, welcher der beiden Spieler nun an der Reihe ist. Mit der Methode ,,writer.flush()" versendet er die Nachricht. Der Host hat sein eigenes Spielfeld mit den einzelnen Feldzeichen.

Damit der Client mit dem selben Feld spielen kann, wird das Feld vom Host zum Client versendet. Diese Methode wird mit einem Char-Array realisiert.

Danach wird die Methode ,,run()" implementiert. Hierbei wartet ein Spieler solange

bis er Verbunden ist. Daraufhin wird der Zug vom Spieler gesetzt. Es wird „Platform.runLater(()" erstellt. Das bedeutet, dass man von dem anderen Thread, in die JavaFx Thread eine Funktion ausführen kann, ohne, dass es zu einem sogenannten freeze, also einem einfrieren kommt. Danach wird ein neuer BufferedReader erstellt. Dieser ist abhängig vom Client und dient dazu Nachrichten lesen zu können. Anschließend wird neuer PrintWriter erstellt, welcher auch abhängig vom Client ist, aber er sich vom BufferedReader darin unterscheidet, dass er die Nachrichten versenden kann, anstatt sie nur zu lesen. Zunächst wird das Feld dem Spieler versendet. Anschließend wird dem Spieler seinen Zug angepasst.

In der Methode „run()" wird außerdem ein „myGui.setLock(true)" implementiert. Diese bewirkt, dass sobald ein Spieler nicht dran ist, für ihn das Klicken gesperrt ist. Das heißt, dass der Spieler während dieser Zeit nichts drücken kann. Sobald einer der Spieler dran ist, darf dieser Spieler insgesamt zwei mal drücken. Mit Hilfe des Reader, wird das „ButtonClicked" gelesen. Der „Tokenizer" erstellt eine verkettete Liste. Mit „next.token" werden immer die nächsten Eingaben eingespeichert. Dabei steht Wert1 für die Spalte und Wert2 für die Zeile. Sobald man beim zweiten Klick angelangt ist, und das „checkifSame" gleich dem Button ist, der vorher gedrückt wurde, dann sollen alle Werte zurückgesetzt werden und der Spieler darf nochmal zwei Button drücken.

Wenn man das erste mal gedrückt hat, werden die Werte von x und y in Wert3 und in Wert4 gespeichert. Dies geschieht aus dem Grund, damit beim nächsten mal Drücken der Button diese beiden miteinander verglichen werden.

Fazit

Wir hatten eine menge Spaß bei der Programmierung des „MemoryGame“.
Wir haben uns zunächst erst einmal zusammengesetzt und besprochen für welches Spiel wir uns entscheiden, welche Methoden für dieses Spiel am besten geeignet wären und wie wir überhaupt vorgehen wollen.
Als wir das abgeschlossen haben, haben wir uns erst einmal daran begeben verschiedene Dinge auszuführen um zu schauen ob dies so funktioniert wie wir uns das erhofft haben. Manchmal mussten wir noch etwas korrigieren aber allgemein sind unsere Ideen oft aufgegangen was uns sehr gefreut hat.
Als unser grober 'Bauplan' stand haben wir mit den Anfängen begonnen. Wir haben uns zwar einzelne Teile aufgeteilt, sodass jeder ein Spezialist war, dennoch waren wir fast immer zusammen und haben dem anderen geholfen.
Für manche Dinge haben wir eher weniger Zeit benötigt für andere wie z B. Der GUI und dem ServerClient haben wir mehr Zeit investiert.
Dennoch haben wir nie dran gedacht das es nicht klappen könnte, da wir eigentlich recht viel Spaß beim nachdenken und zusammenbauen eines Programms hatten.
Also unser Programm stand haben wir uns an sog. Feinheiten begeben wir haben geschaut wo wir manche Dinger vielleicht noch flüssiger oder schöner darstellen können, danach haben wir selber das erste mal an unserem Programm gespielt und waren natürlich sehr stolz.
Das Projekt war im allgemeinen eine sehr gute Übung für uns und hat uns nochmal einen besseren Einblick in die richtige Programmierung gegeben und uns gezeigt das manches weniger kompliziert ist als man denkt.
Wir sind ehrlich; am Anfang dachte wir es wäre uns nicht möglich so etwas jetzt schon zu programmieren, aber je weiter und tiefer man sich mit der Thematik hinein gearbeitet hat, desto mehr haben wir verstanden und auch mehr Selbstvertrauen gefasst. Wir glauben, dass das Projekt auch in Hinsicht auf die Prüfung ein wichtiger Teil ist und besser als jede stumpfe Erklärung ist, wenn
man etwas praktisch beherrscht, dann auch in der Prüfung deshalb hat uns das in der Hinsicht auch mehr Selbstvertrauen gegeben diese zu schaffen.