Projektdokumentation

für ein Projekt der Brühlwiesenschule im Ausbildungsberuf
Fachinformatiker für Anwendungsentwicklung

Thema:

Memophant
—
Das Memory der besonderen Art

Projektplanung und Durchführung:

Armin Chakmehdouz
Bundesamt für Wirtschaft und Ausfuhrkontrolle

Alisha Klein
Statistisches Bundesamt

Paul Onutor
SP-Integration

Auftraggeber:

Peter Grüning
Brühlwiesenschule Hofheim

Ort und Datum:

Hofheim, den 30.08.2012

Inhaltsverzeichnis

1 Projektbeschreibung 3
1.1 Projektbegründung 3
1.2 Projektumfeld 3
1.2.1 Das Projektteam 3
1.2.2 Der Auftraggeber 3
1.2.3 Softwareumgebung 3
2 Ressourcen- und Ablaufplanung 4
2.1 Zeitplanung 4
2.2 Geldplanung 5
2.3 Benötigte Sachmittel 5
3 Fachkonzept 6
3.1 Ziele des Projekts 6
3.2 Spezifikation der Systemfunktionen 6
3.3 Datenbankmodell 6
3.4 Klassendiagramm 9
3.5 Anwendungsfalldiagramm 17
3.6 Aktivitätsdiagramm 18
4 Realisierung 20
4.1 Gestaltung der graphischen Benutzeroberfläche 20
4.2 Realisierung des Servers 26
4.3 Implementierung der Spiellogik 27
4.4 Verknüpfung von Server und Oberfläche 27
5 Tests 28
5.1 Testkonzepte 28
5.2 Testfälle 28
5.3 Testprotokolle 29
6 Projektergebnis 30
6.1 Soll-/Ist- Vergleich 30
6.2 Projektverlauf und Fazit 30
A. Anhang .....
A.1. Lastenheft .....
A.2. Pflichtenheft .....
A.3. Installationsanleitung .....
A.4. Bedienungsanleitung .....
A.5. Testprotokolle .....
A.6. Teamprotokolle .....

1 Projektbeschreibung

Im Rahmen der Ausbildung zum Fachinformatiker Anwendungsentwicklung sollte ein Programm in Anlehnung an das Ravensburger-Spiel „Memory“ entwickelt werden. Da keine Web-Applikation entwickelt werden sollte, wurde die Anwendung als Stand-Alone-Anwendung entwickelt. Als Namen für das Projekt wählten wir „Memophant“.

1.1 Projektbegründung

Die Entwicklung des Spiels ist vorwiegend ein auf das Abschlussprojekt der Ausbildung vorbereitendes Projekt gewesen, das den Teammitgliedern helfen sollte, ein Gespür für die Projektarbeit zu entwickeln. Zu diesem Zweck mussten sich die Teammitglieder streng an die Vorgaben des Auftraggebers und der IHK halten.

1.2 Projektumfeld

1.2.1 Das Projektteam

Rolle / Rollen	Name	Telefon	E-Mail	Bemerkungen
Projektleiterin	Alisha Klein	0611754848	*Alisha.klein@destatis.de* *Alisha.klein@ymail.com*	Statistisches Bundesamt
Protokollant/ Teilnehmer 1	Paul Onutor	017664629101	*paul@onutor.de*	SP Integration GmbH
Teilnehmer 2	Armin Chakmehdouz	0178 20 24 026/ 06196 908 171	*Armin.chakmehdouz-ghasemi@bafa.bund.de* *arminhawk@web.de*	Bundesamt für Wirtschaft und Ausfuhrkontrolle

1.2.2 Der Auftraggeber

Auftraggeber des Projekts war die Brühlwiesenschule in Hofheim im Taunus. Der Vertreter und Ansprechpartner für das Projekt ist Peter Grüning, p.gruening@bws-hofheim.de.

1.2.3 Softwareumgebung

Die Applikation wurde in der objektorientierten Programmiersprache JAVA nach der Vorgabe der Server-Client-Architektur umgesetzt. Der serverseitige Teil des Programms greift auf eine SQLite-Datenbank zu, speichert die zu persisitierenden Daten darin und ruft sie bei Bedarf ab. Die Client-Applikation speichert, wenn keine Verbindung zum Server besteht, die Daten lokal in einer SQLite-Datenbank ab. Die Daten der Anmeldung zur Vorbelegung der Eingabefelder werden in einer Java-Properties-Datei gesichert.

2 Ressourcen- und Ablaufplanung

2.1 Zeitplanung

Der Zeitraum, in welchem das Projekt geplant und entwickelt wurde, begann am 13. August 2012 und endete am 30. August 2012. Die Projektzeit betrug 3 Wochen. Die von der Schule zur Verfügung gestellte Zeit betrug pro Woche 25,5 Zeitstunden. Die Zeitstunden im Projektzeitraum summieren sich auf 76,5 abzüglich 3 Stunden, da am letzten Tag der dritten Woche das Projekt fertig sein musste. Von insgesamt 73,5 Stunden blieben letztendlich noch 66 Stunden übrig – in den anderen Stunden wurde unterrichtet und eine Weiterarbeit am Projekt nicht erlaubt.

Das Team besteht aus 3 Mitgliedern, daher ergab sich eine Anzahl von 198 Personenstunden (PS).

Die Verteilung der Aufgaben auf Personen in Personenstunden wurde wie folgt geplant:

Aufgabe	Zuständig	Beginnt am	Endet am	PS
Implementierung Client-Anwendung	Fr. Klein Hr. Chakmehdouz	13.08.2012	24.08.2012	80
Implementierung Server-Anwendung	Hr. Onutor	13.08.2012	22.08.2012	25
Test Client-Anwendung	Fr. Klein Hr. Chakmehdouz	27.08.2012	29.08.2012	25
Test Server-Anwendung	Hr. Onutor Hr. Chakmehdouz	23.08.2012	29.08.2012	40
Dokumentation	Hr. Onutor	13.08.2012	30.08.2012	15
Präsentation	Fr. Klein	27.08.2012	30.08.2012	13

Die Zeit für die Implementierung der Client-Anwendung wurde wie folgt aufgeteilt:

Aufgabe	Zuständig	Beginnt am	Endet am	PS
Umsetzen der Benutzeroberfläche	Fr. Klein	13.08.2012	17.08.2012	25
Erstellen der Netzwerklogik	Hr. Chakmehdouz	20.08.2012	22.08.2012	17
Entwickeln der Spiellogik	Fr. Klein Hr. Chakmehdouz	13.08.2012	22.08.2012	25

Verbinden von Front- & Backend	Fr. Klein Hr. Chakmehdouz	20.08.2012	24.08.2012	13

Die Entwicklung der Server-Anwendung wurde wie folgt aufgeteilt:

Aufgabe	Zuständig	Beginnt am	Endet am	PS
Kleine Gui-Ausgabe	Hr. Onutor	13.08.2012	13.08.2012	4
Erstellen der Netzwerklogik	Hr. Onutor	14.08.2012	15.08.2012	7
Spielmanagement	Hr. Onutor	16.08.2012	20.08.2012	12
Spielerverwaltung	Hr. Onutor	20.08.2012	22.08.2012	16

2.2 Geldplanung

Die Planung des Projekts lässt sämtliche finanzielle Aspekte außer Acht. Allerdings muss erwähnt werden, dass die Zurverfügungstellung von Räumlichkeiten und Mobiliar der Brühlwiesenschule durch Steuergelder und Zahlungen der Ausbildungsbetriebe honoriert werden.

2.3 Benötigte Sachmittel

Die für das Projekt benötigten Mittel umfassen die üblichen Einrichtungen in einem Büro. Für alle Teammitglieder werden Schreibtische, Stühle, PCs und USB-Speichersticks benötigt.

Von der Schule wurden folgende Sachgegenstände zur Verfügung gestellt:

- Schreibtische
- Stühle
- PCs

Aufgrund der an den PCs ausgeübten Administration durch das Land Hessen und häufiger Überlastung des Netzwerks in der Schule, einigte sich das Team darauf fortan auf privaten Notebooks weiterzuentwickeln. Durch den Zugang zum Internet durch einen WLAN-Hotspot der Schule war es zudem möglich, auf ein Online-Repository zugreifen zu können.

3 Fachkonzept

3.1 Ziele des Projekts

- Entwicklung des Projektes „Memophant“ in Anlehnung an das Ravensburger-Spiel „Memory“
- Entwicklung einer Applikation als Desktop-Anwendung – lokal, oder im Client-Server-Betrieb mit der Möglichkeit mehrere Spieler über einen Server gegeneinander antreten lassen zu können
- Erstellung von Highscorelisten und Persistierung der Anmelde- und Spieldaten in einer Datenbank
- Einbindung einer Chatfunktion
- Einbindung einer programminternen Hilfe mit Spielregeln
- Platformunabhängige, objektorientierte Umsetzung der Anwendung

3.2 Spezifikation der Systemfunktionen

Das Projekt wird parallel auf drei Notebooks und einem PC bearbeitet. Auf allen genannten Komponenten befindet sich das Betriebssystem Windows. Die niedrigste Version des Betriebssystems ist Windows XP, die höchste Windows 7.

Auf den Komponenten wird die Entwicklungsumgebung Eclipse installiert. Alle Teammitglieder sind angehalten, diese Entwicklungsumgebung zu nutzen.

Des Weiteren wird innerhalb des Projektteams ein Repository genutzt, welches über das Programm Turtoise SVN auf dem Webportal www.assembla.com zu erreichen ist. Alle Teammitglieder erhielten zu Beginn des Projekts einen Benutzeraccount für das SVN.

3.3 Datenbankmodell

Das Datenbankmodell des Projekts ist in zwei Parts gesplittet. Zum einen existiert eine Datenbank für alle online gespeicherten Daten, zum anderen eine Datenbank für lokal Gespeichertes.

Abb 1. Datenbankmodell der lokalen Datenbank

Die lokale Datenbank beinhaltet eine Tabelle namens *platzierung*. In dieser Tabelle werden die Spielergebnisse, auch Highscores, gespeichert. Folgende Angaben werden in die Tabelle eingetragen:

Attribut	Funktion
ID	Eindeutige Identifizierung des Highscore-Eintrages
spielerName	Der Name des Spielers
datum	Datum, an dem das Ergebnis erspielt wurde
spieldauer	Dauer des Spiels
spielzüge	Anzahl der Spielzüge, die der Spieler brauchte
spielart	Angabe des Schwierigkeitsgrad: *low*, *middle*, *high*

Der Aufbau der Online-Datenbank ist prinzipiell ähnlich, allerdings wurde die Spielerverwaltung mitberücksichtigt und somit war eine Normalisierung nötig, da mehr als eine Tabelle in der Datenbank enthalten war.

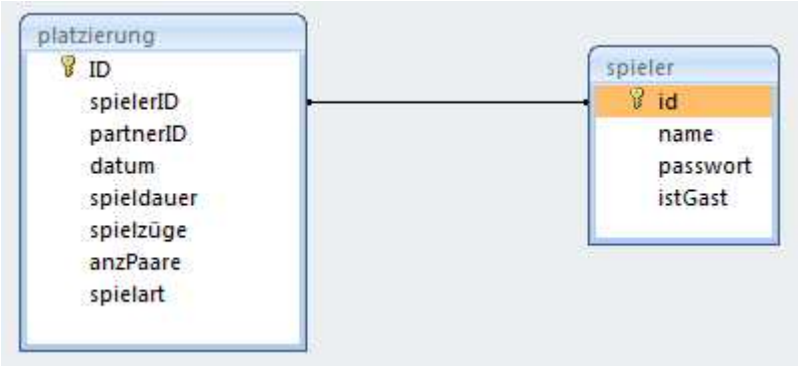


Abb 2. Datenbankmodell der Online-Datenbank

Im Gegensatz zur lokalen Datenbank wurde der Spielername zusammen mit weiteren Attributen in die Tabelle *spieler* ausgelagert. Verknüpft wird die Tabelle *platzierung* mit der Tabelle *spieler* primär über dem Fremdschlüssel *spielerID*, solange es sich um ein Einzelspiel handelte. Wenn ein Partnerspielergebnis gespeichert werden soll, wird das Attribut *partnerID* ebenfalls als Fremdschlüssel mit der Tabelle *spieler* verknüpft.

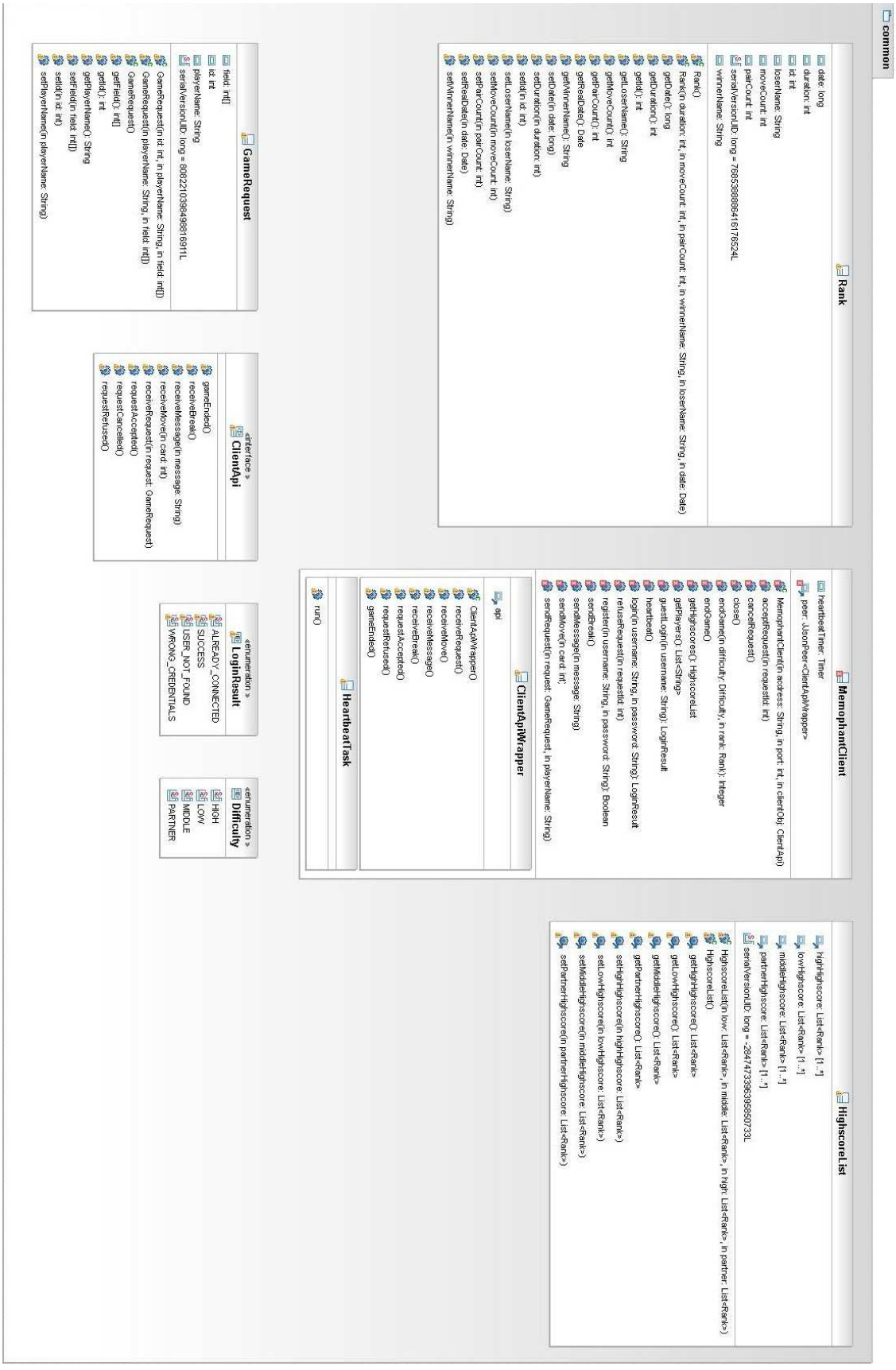
Tabelle *platzierung*:

Attribut	Funktion
ID	Eindeutige Identifizierung des Highscore-Eintrages
spielerID	ID des 1. Spieler
partnerID	ID des 2. Spielers
datum	Datum, an dem das Ergebnis erspielt wurde
spieldauer	Dauer des Spiels
spielzüge	Anzahl der Spielzüge, die der Spieler brauchte
spielart	Angabe des Schwierigkeitsgrad: *low*, *middle*, *high*

Tabelle *spieler*:

Attribut	Funktion
ID	Eindeutige Identifizierung des Spielers
name	Der Name des Spielers
passwort	Verschlüsseltes Accountpasswort, wenn Spieler kein Gast ist
istGast	Enthält *wahr*, wenn der Spieler als Gast angemeldet ist. Ansonsten *falsch*.

3.4 Klassendiagramm

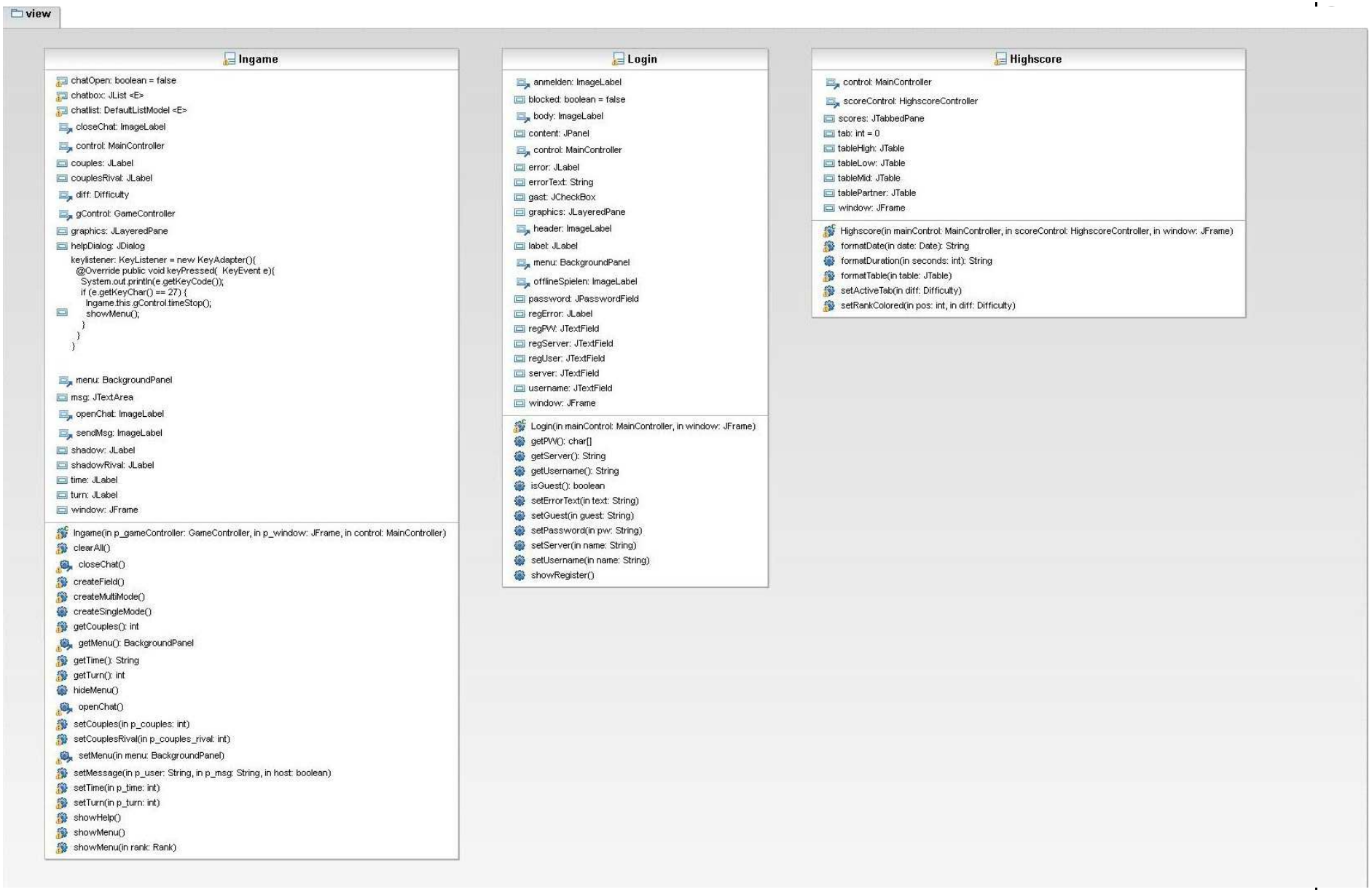


controller
GameController
blocked: boolean = false
control: MainController
diff: Difficulty
field: Field
finished: boolean = false
game: Ingame
guestplayer: Player
hostplayer: Player
keypressedaction: KeyPressedAction
openCard1: Card = null
openCard2: Card = null
rank: Rank
singlemode: boolean
started: boolean = false
time: Int
timeStop: boolean = false
window: JFrame
winner: Player
GameController(in p_mainController: MainController, in p_window: JFrame, in guest: User)
GameController(in p_mainController: MainController, in p_window: JFrame, in host: User, in field: Field)
calcWinner(): Player
cardClicked(in p_card: Card): ReplyToMove
changePlayingUser()
compareCards(): ReplyToMove
generateField()
getControl(): MainController
getDiff(): Difficulty
getField(): Field
getGame(): Ingame
getGuestplayer(): Player
getHostplayer(): Player
getLoser(): Player
getLosserName(): String
getPlayingPlayer(): Player
getPlayingUser(): User
getWinner(): Player
getWinnerExist(): ReplyToMove
goToHighscore()
isBlocked(): boolean
isSinglemode(): boolean
listenToTheKeys()
move(in card: Card): ReplyToMove
move(in card: Card, in ownMove: boolean): ReplyToMove
newGame()
quitGame()
removeKey(Listener)
setBlocked(in blocked: boolean)
setBlockedTime(in sec: int)
setDiff(in diff: Difficulty)
setField(in field: Field)
setGuestMessage(in p_msg: String)
setGuestplayer(in guestplayer: Player)
setHostplayer(in hostplayer: Player)
setMessage(in p_msg: String)
setSinglemode(in singlemode: boolean)
setWinner(in winner: Player)
showBreakMenu()
start()
start(in field: Field)
startTimer()
timeContinue()
timeStop()
turnCardsBack()
MainController
chosenPlayer: String
connectControl: ConnectionController
db: Database
gameControl: GameController
gameRequest: GameRequest
l: Login
m: Menu
midPW: String = null
o: Options
online: boolean
player: Player
possibleGuest: String
s: SaveProps
scoreControl: HighscoreController
user: User
window: JFrame
MainController()
askForRequestTaking(in request: GameRequest): boolean
createMID(in s: String): String
createSendRequest(in chosenPlayer: String)
createWindow()
gameRefused()
getChosenPlayer(): String
getConnectControl(): ConnectionController
getGameControl(): GameController
getGameRequest(): GameRequest
getImage(in p_file: String): BufferedImage
getPlayer(): Player
getPossibleGuest(): List<String>
getProp(in p_key: String): String
getResource(in p_file: String): InputStream
getScreenHeight(): int
getScreenWidth(): int
getUser(): User
isOnline(): boolean
playOffline()
playOnline()
quitGame(in diff: Difficulty, in rank: Rank)
register(in p_user: String, in p_pw: String, in server: String): boolean
sendMessage(in p_msg: String)
setConnectControl(in connectControl: ConnectionController)
setGameControl(in gameControl: GameController)
setGameRequest(in gameRequest: GameRequest)
setOnline(in p_online: boolean)
setPlayer(in player: Player)
setPossibleGuest(in possibleGuest: String)
setProp(in p_key: String, in p_value: String)
setUser(in user: User)
showHighscore(in r: Rank)
showMenu()
showMessage(in p_msg: String)
showOptions()
startGame(in p_type: ViewType, in request: GameRequest)
windowClear()
windowRepaint()
windowValidate()
ConnectionController
connected: boolean = false
control: MainController
memoClient: MemophantClient
user: User
ConnectionController(in user: User, in control: MainController, in serveradress: String)
cancelRequest()
endgame(in diff: Difficulty, in rank: Rank)
gameEnded()
getControl(): MainController
getHighscore(): HighscoreList
getMemoClient(): MemophantClient
getPlayer(): List<String>
getUser(): User
guestLogin(in username: String): LoginResult
isConnected(): boolean
login(in username: String, in password: String): LoginResult
receiveBreak()
receiveMessage(in message: String)
receiveMove(in card: int)
receiveRequest(in request: GameRequest)
register(in username: String, in password: String): boolean
requestAccepted()
requestCancelled()
requestRefused()
sendBreak()
sendMessage(in message: String)
sendMove(in card: int)
sendRequest(in request: GameRequest, in playerName: String)
setControl(in control: MainController)
setMemoClient(in memoClient: MemophantClient)
setUser(in user: User)

controller
HighscoreController
control: MainController
db: Database
high: List<Rank> [1..*]
highscores: HighscoreList
low: List<Rank> [1..*]
middle: List<Rank> [1..*]
partner: List<Rank> [1..*]
rank: Rank
score: Highscore
window: JFrame
HighscoreController(in p_mainController: MainController, in p_window: JFrame, in p_r: Rank)
getHigh(): List<Rank>
getLow(): List<Rank>
getMiddle(): List<Rank>
getPartner(): List<Rank>
getPosition(): int
getWindow(): JFrame
setHigh(in high: List<Rank>)
setLow(in low: List<Rank>)
setMiddle(in middle: List<Rank>)
setPartner(in partner: List<Rank>)
setWindow(in window: JFrame)
MouseCardClickAction
card: Card
gc: GameController
MouseCardClickAction(in p_control: GameController, in p_card: Card, in p_label: ImageLabel)
mouseReleased(in e: MouseEvent)
Database
connection: Connection
dateFormatter: DateFormat
syncObj: Object = new Object()
Database()
Dispose()
init()
getHighscore(in diff: Difficulty): List<Rank>
saveRank(in diff: Difficulty, in rank: Rank)
KeyPressAction
gc: GameController
KeyPressAction(in p_gameController: GameController)
dispatchKeyEvent(in e: KeyEvent): boolean

model
Card
cardCovers: ArrayList<BufferedImage>
cardback: String
cardcover: String
index: int
turned: boolean
value: int
view: ImageLabel
Card(in index: int, in wert: int, in cardcover: String, in cardback: String)
Card(in index: int, in wert: int, in cardCovers: ArrayList<BufferedImage>, in cardback: String)
equals(in card: Card): boolean
getIndex(): int
getPathDiscoverd(): String
getValue(): int
getView(): ImageLabel
isTheSameAs(in card: Card): boolean
isTurned(): boolean
setImage()
setIndex(in index: int)
setPathDiscoverd(in pathDiscoverd: String)
setTurned(in turned: boolean)
setValue(in value: int)
setView(in view: ImageLabel)
toString(): String
turn()
Field
cardCovers: ArrayList<BufferedImage> = new ArrayList<BufferedImage>()
cardback: String
cardcover: String
cards: ArrayList<Card> = new ArrayList<Card>()
control: MainController
Field(in control: MainController, in size: int)
Field(in p_control: , in cards:)
countUnfoundPairs(): int
getCardByIndex(in p_index: int): Card
getCardByPosition(in p_pos: int): Card
getCards(): ArrayList<Card>
getTurnedCard(): Card
getValues(): int[]
loadCardCovers()
mixup()
setCards(in cards: ArrayList<Card>)
toString(): String
turnCard(in card: Card)
User
guest: boolean = false
online: boolean = false
password: String
username: String
User(in p_username: String)
User(in p_username: String, in p_password: String, in p_guest: boolean)
getPassword(): String
getUsername(): String
isGuest(): boolean
isOnline(): boolean
setGuest(in p_guest: boolean)
setOnline(in p_online: boolean)
setPassword(in p_password: String)
setUsername(in p_username: String)
toString(): String
Player
movecount: int
pair: int
turn: boolean
user: User
Player(in user: User)
addMovecount()
addPair()
getMovecount(): int
getPair(): int
getUser(): User
isTurn(): boolean
reset()
setMovecount(in movecount: int)
setPair(in pair: int)
setTurn(in turn: boolean)
setUser(in user: User)
SaveProps
p: Properties
path: String
SaveProps(in p_path: String)
getProp(in p_key: String): String
save()
writeProp(in p_key: String, in p_value: String)

enums
«enumeration»
ReplyToMove
FIRST_CARD
GEWONNEN
NOT_ALLOWED
PAIR_FOUND
PAIR_NOT_FOUND
UNENTSCHIEDEN
«enumeration»
ViewType
HIGHSCORE
LOGIN
MENU
MULTI
SINGLE
test
TestClient
client: MemophantClient
clientId: int
TestClient(in clientId: int, in host: String, in port: int)
gameEnded()
getClient(): MemophantClient
receiveBreak()
receiveMessage(in message: String)
receiveMove(in card: int)
receiveRequest(in request: GameRequest)
requestAccepted()
requestCancelled()
requestRefused()
TestApplication
main(in args: String[])
renderer
RowColorRenderer
row: int
RowColorRenderer(in p_row: int)
getTableCellRendererComponent(in table: JTable, in value: Object, in isSelected: boolean, in hasFocus: boolean, in row: int, in column: int): Component
ChatBoxRenderer
getListCellRendererComponent(in list: JList, in value: Object, in index: int, in isSelected: boolean, in cellHasFocus: boolean): Component


view
Ingame
chatOpen: boolean = false
chatbox: JList <E>
chatlist: DefaultListModel <E>
closeChat: ImageLabel
control: MainController
couples: JLabel
couplesRival: JLabel
diff: Difficulty
gControl: GameController
graphics: JLayeredPane
helpDialog: JDialog
keylistener: KeyListener = new KeyAdapter(){
@Override public void keyPressed(KeyEvent e){
System.out.println(e.getKeyCode());
if (e.getKeyChar() == 27) {
Ingame.this.gControl.timeStop();
showMenu();
}
}
}
menu: BackgroundPanel
msg: JTextArea
openChat: ImageLabel
sendMsg: ImageLabel
shadow: JLabel
shadowRival: JLabel
time: JLabel
turn: JLabel
window: JFrame
Ingame(in p_gameController: GameController, in p_window: JFrame, in control: MainController)
clearAll()
closeChat()
createField()
createMultiMode()
createSingleMode()
getCouples(): int
getMenu(): BackgroundPanel
getTime(): String
getTurn(): int
hideMenu()
openChat()
setCouples(in p_couples: int)
setCouplesRival(in p_couples_rival: int)
setMenu(in menu: BackgroundPanel)
setMessage(in p_user: String, in p_msg: String, in host: boolean)
setTime(in p_time: int)
setTurn(in p_turn: int)
showHelp()
showMenu()
showMenu(in rank: Rank)
Login
anmelden: ImageLabel
blocked: boolean = false
body: ImageLabel
content: JPanel
control: MainController
error: JLabel
errorText: String
gast: JCheckBox
graphics: JLayeredPane
header: ImageLabel
label: JLabel
menu: BackgroundPanel
offlineSpielen: ImageLabel
password: JPasswordField
regError: JLabel
regPW: JTextField
regServer: JTextField
regUser: JTextField
server: JTextField
username: JTextField
window: JFrame
Login(in mainControl: MainController, in window: JFrame)
getPW(): char[]
getServer(): String
getUsername(): String
isGuest(): boolean
setErrorText(in text: String)
setGuest(in guest: String)
setPassword(in pw: String)
setServer(in name: String)
setUsername(in name: String)
showRegister()
Highscore
control: MainController
scoreControl: HighscoreController
scores: JTabbedPane
tab: int = 0
tableHigh: JTable
tableLow: JTable
tableMid: JTable
tablePartner: JTable
window: JFrame
Highscore(in mainControl: MainController, in scoreControl: HighscoreController, in window: JFrame)
formatDate(in date: Date): String
formatDuration(in seconds: int): String
formatTable(in table: JTable)
setActiveTab(in diff: Difficulty)
setRankColored(in pos: int, in diff: Difficulty)

view
AcceptGameRequest
control: MainController
graphics: JLayeredPane
menu: BackgroundPanel
player: String
window: JFrame
AcceptGameRequest(in control: MainController, in window: JFrame, in playerName: String)
getControl(): MainController
getWindow(): JFrame
setControl(in control: MainController)
setWindow(in window: JFrame)
Menu
buttonsblocked: boolean = false
control: MainController
graphics: JLayeredPane
list: JList<String>
menu: BackgroundPanel
window: JFrame
Menu(in mainControl: MainController, in window: JFrame)
showLobby()
showWaitingDialog()
Options
comboBox: JComboBox<String>
control: MainController
diff: JSlider
window: JFrame
Options(in mainControl: MainController, in window: JFrame)
ImageLabel
serialVersionUID: long = -7904078387519871 49L
ImageLabel(in path: String)
setIcon(in path: String)
BackgroundPanel
image: Image
BackgroundPanel(in path: String)
paintComponent(in g: Graphics)

server
Session
TIMEOUT_HEARTBEAT: long = 5000 [0..1]
db: Database
lastHeartbeat: long
logger: Logger = Server.logger
nextRequestId: AtomicInteger = new AtomicInteger(1)
partner: Player
peer: JsonPeer<Session>
player: Player
requests: Map<Long,String> = Collections.synchronizedMap(new HashMap<Long,String>())
sessions: List<Session> = Collections.synchronizedList(new ArrayList<Session>()) [1..*]
Session(in socket: Socket)
acceptRequest(in requestId: long)
cancelRequest()
close()
endGame()
endGame(in difficultyString: String, in rankString: String): int
findSession(in player: Player): Session
getHighscores(): String
getPartner(): Player
getPeer()
getPlayer(): Player
getPlayers(): List<String>
getSessions(): List<Session>
getTIMEOUT_HEARTBEAT(): long
guestLogin(in username: String): String
heartbeat()
isAlive(): boolean
isPlaying(): boolean
login(in username: String, in password: String): String
refuseRequest(in requestId: long)
register(in username: String, in password: String): boolean
removeSession(in session: Session)
sendBreak()
sendMessage(in message: String)
sendMove(in card: long)
sendRequest(in requestString: String, in playerName: String)
setPartner(in player: Player)
setTIMEOUT_HEARTBEAT(in TIMEOUT_HEARTBEAT: long)
Player
guest: boolean
id: int
password: String
username: String
Player()
Player(in id: int, in username: String, in password: String, in guest: boolean)
getId(): int
getPassword(): String
getUsername(): String
isGuest(): boolean
setGuest(in guest: boolean)
setId(in id: int)
setPassword(in password: String)
setUsername(in username: String)
Server
exit: boolean
logger: Logger = Logger.getLogger(Server.class.getCanonicalName())
port: int
ttread: Thread
Server(in port: int)
checkSessions()
main(in args: String[])
run()
start()
stop()
Database
connection: Connection
dateFormatter: DateFormat
instance: Database
syncObj: Object = new Object()
Database()
Dispose()
Init()
findPlayer(in username: String): Player
getHighscore(in difficulty: Difficulty): List<Rank>
getInstance(): Database
savePlayer(in player: Player)
saveRank(in difficulty: Difficulty, in rank: Rank)
ServerWindow
container: Container
logList: JList<String>
logListModel: DefaultListModel<String>
serialVersionUID: long = 8720902044020410269L
ServerWindow()
addHandlers()

3.5 Anwendungsfalldiagramm

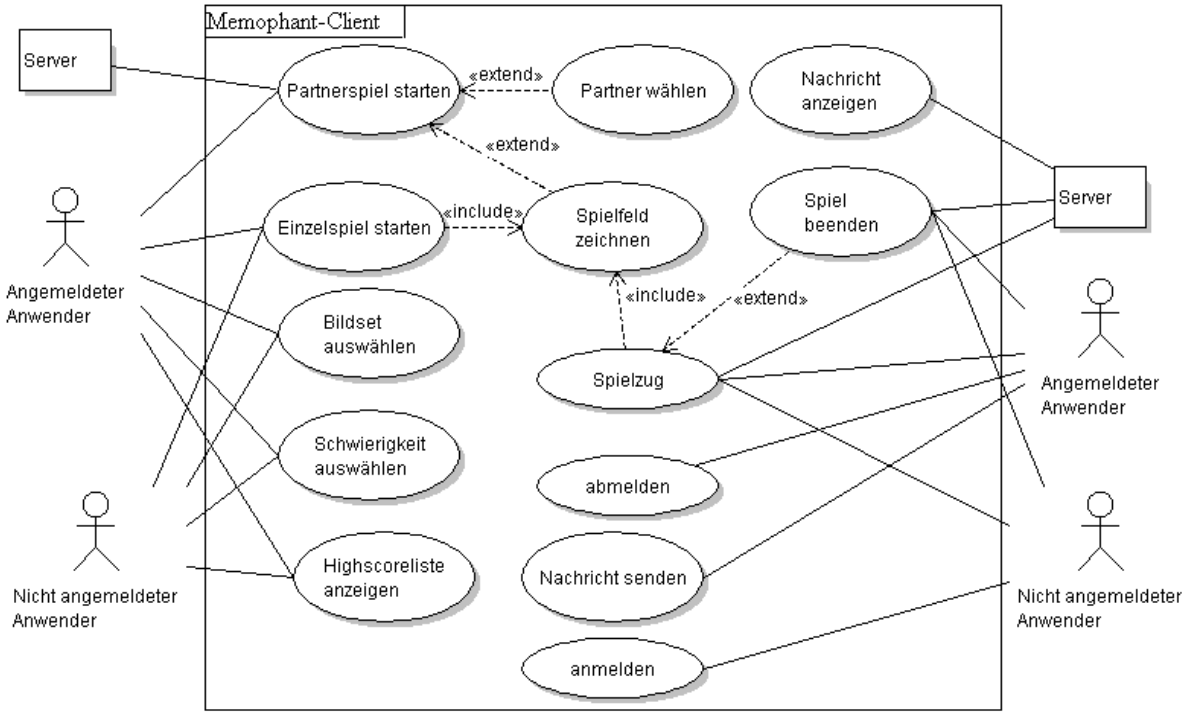


Abb 3. Use-Case-Diagramm der Client-Anwendung

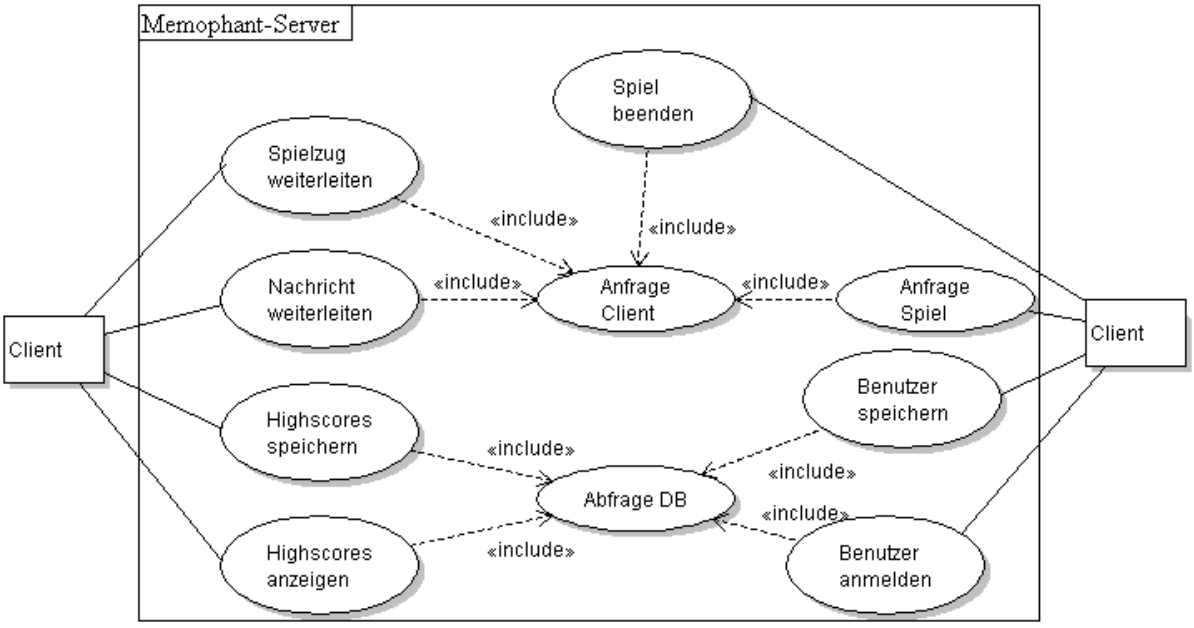


Abb 4. Use-Case-Diagramm der Server-Anwendung

3.6 Aktivitätsdiagramm

Nach dem Starten der Anwendung hat der Benutzer zwei Möglichkeiten – er kann sich an einem Server anmelden bzw. registrieren oder er kann die Anwendung ohne Anmeldung benutzen. Anschließend muss er im Menü zwischen den folgenden Punkten wählen:

1. Spieloptionen
2. Highscoreliste
3. Einzelspiel

Wenn sich der Benutzer an einem Server angemeldet hat, besteht zusätzlich folgende Möglichkeit:

4. Partnerspiel

Unter dem Punkt Spieloptionen kann der Benutzer den Schwierigkeitsgrad des Spiels sowie die Motive des Kartendecks wählen.

In der Highscoreliste stehen alle Spielergebnisse sortiert nach Erfolg des jeweiligen Spiels.

Klickt der Benutzer auf das Einzelspiel, startet sich das Spiel in der eingestellten Schwierigkeitsstufe.

Wählt der angemeldete Benutzer ein Partnerspiel, so wird er zunächst aufgefordert, einen Partner zu wählen und ihm eine Spielanfrage zu senden. Nimmt der Partner das Spiel an, beginnt das Spiel in der höchsten Schwierigkeitsstufe. Während dem Spiel haben beide Spieler die Möglichkeit miteinander zu chatten.

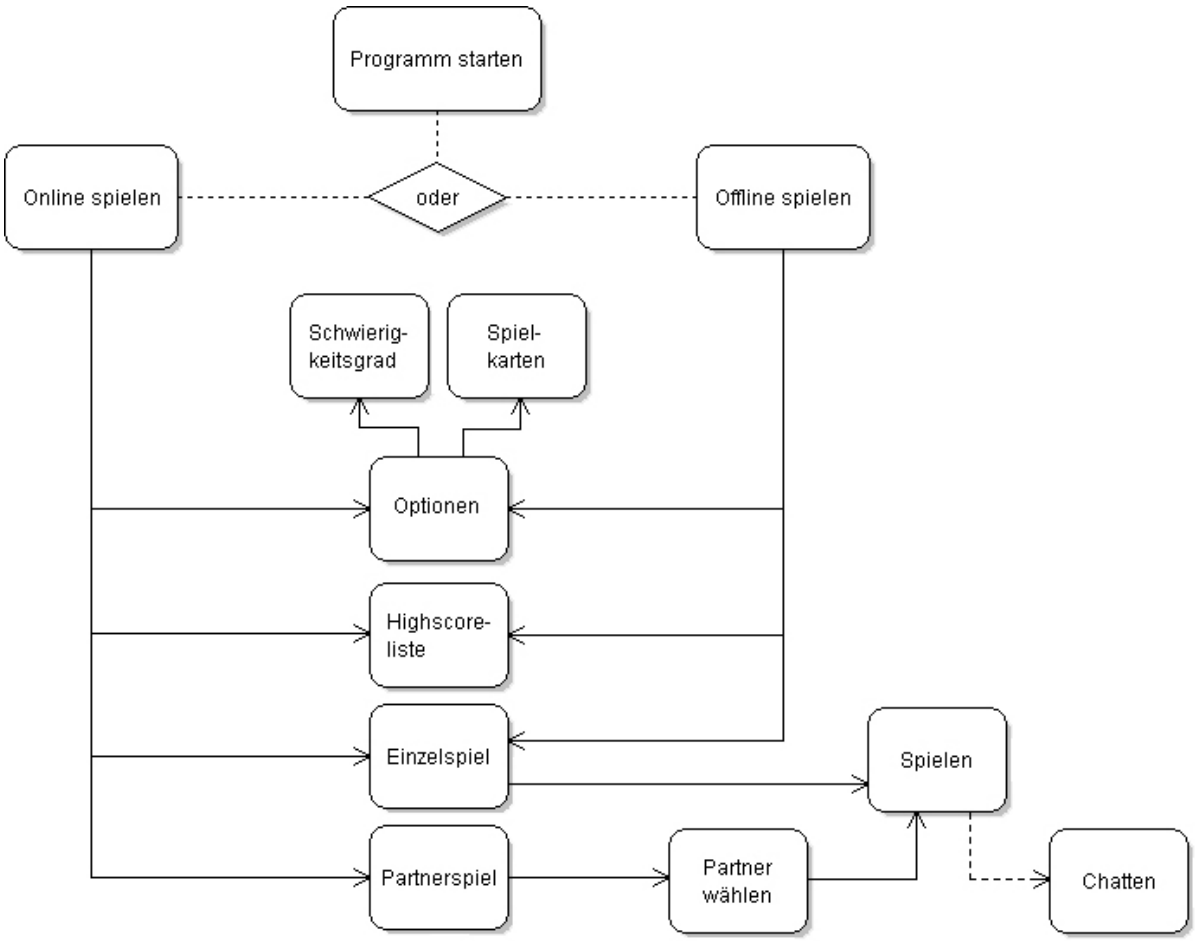


Abb 3. Aktivitätsdigramm

4 Realisierung

4.1 Gestaltung der graphischen Benutzeroberfläche

Die graphische Benutzeroberfläche wurde so konzipiert, dass sie den Anwender zum Spielen animiert. Dies geschieht zunächst durch ein unaufdringliches Farbschema, welches in ruhigen und kühlen Farbbereichen gehalten wurde. Dazu wurden in verschiedenen Nuancen Kontraste integriert, um das Abheben bestimmter Schriftsätze oder Elemente im Spiel zu verdeutlichen. Durch die Integrität der Kontraste im Farbschema ist die Wirkung der Oberfläche sehr subtil – verleitet jedoch zur Auseinandersetzung mit den gegebenen Möglichkeiten.

Weiterhin wurde sehr viel Wert auf Benutzerfreundlichkeit gelegt. Durch das Verändern des Cursors beim Bewegen über bestimmte Elemente, wie z.B. Buttons wird dem Benutzer dezent vermittelt, dass hier Aktionen, z.B. in Form eines Klicks möglich sind. Zudem wurden bestimmte Tasten als sogenannte Hotkeys angelegt. In einem Spiel kann der Benutzer auf die weit verbreitete und anerkannte Pausetaste ESC drücken - ein Pausemenü erscheint und das Spiel wird unterbrochen. Drückt er ein weiteres Mal ESC verschwindet das Pausemenü und das Spiel wird fortgeführt. Ebenso verhält es sich bei den Eingaben in die Textfelder der Loginseite. Drückt ein Benutzer nach Eingabe die Enter-Taste, so wird eine Aktion, wie z.B. das Anmelden, ausgeführt.

Mithilfe einer lokalen Datei werden temporäre Angaben, wie der Benutzername, das verschlüsselte Passwort und die Server-IP-Adresse gespeichert und beim nächsten Aufruf der Loginseite in die entsprechenden Textfelder voreingetragen. Somit bleibt dem Benutzer das immer wieder erneute Eintippen von Daten erspart. Das Spielvergnügen steht im Vordergrund!

Abb 4. Loginbereich

Loginbereich: Im Loginbereich der Anwendung gibt es für den Benutzer zwei deutlich sichtbare Optionen. Er kann sich anmelden, oder offline spielen. Wenn er sich anmelden möchte, muss er zuvor seine Daten in die dafür vorgesehen Felder eintragen. Ohne einen eingetragenen Account kann er sich als Gast eintragen oder sich registrieren lassen. Über den Link auf der Oberfläche gelangt der Anwender zum Registrierungsformular.

Abb 5. Hauptmenü eines Spielers im Offline-Modus

Nach erfolgreicher Anmeldung, Registrierung oder auch nach dem Drücken des Offline-Spielen-Buttons gelangt der Anwender ins Hauptmenü. Die erste Möglichkeit eines jeden Spielers ist es, ein Einzelspiel zu beginnen. Die zweite Möglichkeit ist nicht für jeden Spieler zugänglich. Ein Spieler im Offline-Modus wird auf dezente Art darauf hingewiesen, dass er kein Partnerspiel beginnen kann. Fährt er mit der Maus über den ausgegrauten Button, so erfährt er, dass nur im Online-Modus mit anderen Spielern spielen kann. Ein angemeldeter Spieler erfährt in diesem Punkt keine Einschränkung.

Neben der Möglichkeit ein Spiel zu beginnen, kann der Spieler auch Einstellungen für den Einzelspielermodus ändern. Dazu zählen z.B. der Schwierigkeitsgrad sowie das Kartenmotiv.

Weiterhin kann der Spieler seine errungenen Erfolge in der Highscoreliste begutachten.

Abb 6. Einzelspiel auf Schwierigkeitsstufe leicht

Sobald ein Spiel gestartet wird, kann der Spieler die verdeckten Karten in der Anzahl gemäß seiner gewählten Schwierigkeitsstufe sehen. Die von ihm gewählte Schwierigkeitsstufe steht im oberen Spielbereich unterhalb der Spielzeit. Die Spielzeit wird ab dem ersten Spielzug gestartet. Alle Züge werden mitgezählt, ebenso wird die Anzahl der aufgedeckten Pärchen auf der rechten Seite im oberen Spielbereich festgehalten. Die Anzahl der aufgedeckten Pärchen liegt über dem Bild einer verdeckten Karte, um dem Spieler zu symbolisieren, dass es hier um die Anzahl von Paaren geht.

Abb 7. Lobby mit allen Spielern auf einem Server

Wählt der Anwender den Menüpunkt Partnerspiel aus, so gelangt er nicht direkt in ein Spiel, sondern es erscheint die Lobby. Als Lobby wird die Liste bezeichnet, in der alle Anwender aufgelistet werden, die am Server erfolgreich angemeldet wurden und verfügbar sind. Verfügbar bedeutet, dass der Anwender sich im Hauptmenü oder in einem Einzelspiel befindet. Anfragen an Spieler in einem laufenden Partnerspiel sind nicht möglich.

Abb 8. Multiplayerspiel mit Chatoption

Hat ein angefragter Anwender zugestimmt, öffnet sich das Spielfeld im Partnermodus. Im Partnermodus gibt es nur eine Spielfeldgröße. Die Spieler sind abwechselnd an der Reihe, wenn keine Paare aufgedeckt werden. Deckt ein Spieler ein Pärchen aus, so darf er noch einmal. Während dem Spiel können die Spieler miteinander über einen Chat kommunizieren. Der Chat öffnet sich über einen Button, der unauffällig auf der rechten Seite am Spielfeldrand platziert wurde. Der Button wurde mit Absicht nicht sehr auffällig platziert, da eine eifrige Kommunikation sich negativ auf die Spielzeit auswirkt, welche das Endergebnis im Highscore negativ beeinflussen kann.

Wenn ein Spieler eine Nachricht vom Partner erhält, öffnet sich der Chat automatisch um den Spieler über die empfangene Nachricht zu informieren.

Abb 8. Gewonnenes Multiplayerspiel

Der Spieler mit den meisten Pärchen erhält die Nachricht seines Sieges, ebenso erhält der Spieler mit den wenigeren Pärchen Auskunft über seine Niederlage. Beiden erscheint das Endmenü, in welchem die Möglichkeit angeboten wird, ein weiteres Spiel mit dem selben Partner zu spielen. Weiterhin kann das Ergebnis des Spiels auch in den Highscorelisten begutachtet werden oder man gelangt über den entsprechenden Button zurück ins Hauptmenü.

4.2 Realisierung des Servers

Der Server ist dafür zuständig die Verbindung zwischen zwei Spielern herzustellen und zu verwalten. Spieler müssen sich vorher beim Server anmelden bzw. registrieren falls sie in der Datenbank noch nicht vorhanden sind. Sollte der Spieler sich als Gast anmelden wollen dann muss er einen Namen wählen welcher noch nicht durch einen anderen Spieler registriert wurde.

Die registrierten Spieler werden in einer SQLite-Datenbank gespeichert. Die Verbindung zur Datenbank wird über eine Java-Komponente hergestellt. Einfache Datenklassen werden als Austauschformat zwischen des Datenbankadapters und der Serverlogik benutzt.

Dieser Datenbankadapter beinhalten auch Methoden zum Suchen eines bestimmten Spielers und auslesen der sortierten Highscorelisten.

Über das JSON-RPC System werden Daten zwischen dem Server und der Clients ausgetauscht. Das JSON-Format wird dabei benutzt um Daten menschenlesbar zu übertragen. Das RPC-Protokoll endscheidet anhand der empfangenen Daten welche Methoden auf der jeweiligen Verbindungsseite aufgerufen werden müssen. Das Ergebnis der Methoden wird anschließend zurück an den Sender gesendet.

Der Server lauscht per TCP-Socket auf dem Port 12345 und wartet bis Clients sich verbinden möchten. Sollte ein Client eine Verbindung herstellen, dann wird für diesen Client eine neue Sitzung kreiert und diese in die globale Sitzungsliste eingetragen. Der Client hat nun die Aufgabe im Abstand von 4 Sekunden ein sogenanntes „Heartbeat“-Signal an den Server zu senden. Sollte nach 5 Sekunden für diesen Client kein „Heartbeat“-Signal am Server angekommen sein, dann wird die Sitzung geschlossen und aus der Sitzungsliste gelöscht.

Ein verbundener und eingeloggter Spieler kann eine Liste aller vorhandenen und aktuell nicht spielenden Spieler vom Server anfordern und daraufhin eine Spielanfrage an einen dieser Spieler senden.
Der Server generiert nun eine interne ID und leitet die Anfrage sowie die ID an die jeweilige Sitzung des anderen Spielers weiter. Der angefragte Spieler wird dann aufgefordert die Anfrage anzunehmen oder abzulehnen. Das Ergebnis muss dann gemeinsam mit der ID an den Server zurück gesendet werden.

Wenn die Anfrage angenommen wurde wird in der Sitzung beider Partien markiert das ein Spiel besteht. Die Spieler können sich nun gegenseitig Spielzüge zuschicken. Sollte ein Spieler das laufende Spiel beenden, wird dem anderen Spieler dies mitgeteilt und die Markierung in den Sitzungen wird entfernt. Gleiches gilt für ein gewonnenes Spiel, jedoch wird dem Server dann auch das Ergebnis des Spiels zugesendet, welches vom Server dann in die Datenbank eingetragen wird.

4.3 Implementierung der Spiellogik

Die Spiellogik wurde komplett in die Clientanwendung integriert, da der Client auch ohne den Server lauffähig sein sollte. Sie wurde für das Spielen von Einzel- und Partnerspielen ausgelegt und kann im Fall eines Partnerspiels auch vom Server aus angestoßen werden. Vom Server wird nur der Wert der vom Spieler umgedrehten Karte übergeben. Dadurch wird bei jedem Zug in einem Partnerspiel die Spiellogik auf beiden Clients ausgeführt. Da die Clients identisch sind, ist das identische Ergebnis der Spielzüge gewährleistet.

4.4 Verknüpfung von Server und Oberfläche

Bei der Umsetzung des Projekts wurden streng die Vorgaben des Model-View-Controller-Prinzips eingehalten.

Zur Verknüpfung von Server und Oberfläche wurden also Controller geschrieben, die gegen die Schnittstellen des Servers programmiert wurden. Diese Controller beinhalten ebenfalls die Spiellogik und nehmen die Spielzüge entgegen, die von der Oberfläche angestoßen werden. Sie leiten den Spielzug über den Server an den anderen Spieler weiter und führen danach damit die Spiellogik aus. Beim Ende des Spiels erstellen sie einen Highscoreeintrag und übermitteln diesen an den Server, welcher ihn in die Datenbank einträgt oder bei lokaler Speicherung an einen Controller, der ihn in eine lokale Datenbank einträgt.

Auch Anfragen, die von der Oberfläche an den Server beziehungsweise die lokale Datenbank gehen, wie zum Beispiel beim Login eines Spielers, bei der Anzeige aller Highscores, bei der Auswahl der verfügbaren Spieler auf dem Server etc. geht die Anfrage von der Oberfläche an den entsprechenden Controller, der diese gegebenenfalls an den Server weiterleitet.

5 Tests

5.1 Testkonzepte

Während der Entwicklung der einzelnen Funktionen der Anwendungen wurden regelmäßig Whiteboxtests durchgeführt. Es wurden kleine Testprogramme unter Einsichtnahme des Quellcodes geschrieben, die die Funktionen mit bestimmten Parametern ausführten und das Ergebnis auf der Konsole ausgaben, sodass deren Richtigkeit überprüft werden konnte.

Im Anschluss an die Programmierung der Grundfunktionen wurden überwiegend Blackboxtests durchgeführt und die dabei aufgetretenen Fehler wurden behoben.

5.2 Testfälle

Folgende Fälle wurden getestet:

- Das Starten des Servers
- Das Starten des Hauptmenüs im Offlinemodus
- Die Anmeldung und Registrierung eines Spielers
    - Die Anmeldung eines Spielers als Gast
    - Fehlerhafte Anmeldedaten (Name, Passwort oder Server)
    - Fehlerhafte Registrierungsdaten (Name, Passwort oder Server)
    - Speicherung der Anmeldedaten und Vorbelegung bei Neustart
- Das Starten eines Einzelspiels
- Das Starten eines Partnerspiels (nur im Onlinemodus)
    - Vorherige Auswahl eines Partners
        - Nur mögliche Spielpartner anzeigen
        - Anfrage wird auf anderer Seite angezeigt
        - Bei Annahme des Gegners Spielstart
        - Nur eine Anfrage möglich
        - Verneinung des Spiels möglich
- Die Auswahl der Spieloptionen
    - Speicherung der Spieloptionen und Vorbelegung
- Das Öffnen der Highscorelisten
    - Unterschiedliche Schwierigkeitsstufen in unterschiedlichen Listen
    - Bei Onlinemodus wird Partnerliste angezeigt
    - Korrekte Sortierung der Datensätze in Einzelspiel- und Partnerspiellisten
    - Zurück zum Hauptmenü möglich
- Der korrekte Ablauf eines Einzelspiels
    - Züge, Paare, Zeit richtig zählen
    - Paare aufgedeckt lassen, falsche Karten wieder zudecken
- Der korrekte Ablauf eines Partnerspiels
    - Züge, eigene Paare und Gegnerpaare, Zeit richtig und synchron zählen
    - Abwechselndes Aufdecken der Karten außer bei Pärchen

    - Paare und Einzelkarten auf beiden Seiten aufgedeckt
    - Nur Aufdecken vom spielenden Spieler zulässig
- Pausieren des Spiels
    - Pausieren und Fortsetzen synchron zum Gegner (Partnerspiel)
    - Spiel lässt sich fortsetzen
    - Spiel lässt sich abbrechen
        - Zurück zum Hauptmenü
        - Keine Eintragung in Highscores
        - Starten neuer Spiele möglich
        - Spiel wird auch beim Gegner abgebrochen (Partnerspiel)
    - Spielhilfe kann aufgerufen werden
- Das korrekte Ende eines Einzelspiels
    - Alle Karten aufgedeckt
    - Endmenü anzeigen
    - Eintragung des Ergebnisses in die Datenbank (online über Server, offline lokal)
- Das korrekte Ende eines Partnerspiels
    - Alle Karten aufgedeckt
    - Gewinner und Verlierer aufzeigen
    - Endmenü anzeigen
    - Eintragung der Highscores in die Datenbank über den Server (nur einmal)
- Korrekte Funktionen des Endmenü
    - Erneutes Spielen möglich (bei Partnerspiel erneute Anfrage an anderen Spieler)
    - Aufrufen der Highscores möglich
    - Zurück zum Hauptmenü möglich
- Chatfunktionen
    - Öffnen des Chats bei eingehender Nachricht
    - Senden und Empfangen der Nachrichten
    - Schließen des Chats bei Beenden des Spiels

5.3 Testprotokolle

Die durchgeführten Tests und die Behebung der dabei aufgetretenen Fehler konnten die Robustheit und Korrektheit des Programms erheblich verbessern. Auch die Darstellung wurde aus Gründen der besseren Benutzerfreundlichkeit im Laufe der Tests bei Bedarf angepasst.

Ein ausführliches Testprotokoll der durchgeführten Tests kann im Anhang A.5. eingesehen werden.

6 Projektergebnis

6.1 Soll-/Ist- Vergleich

Das Projektziel, die Entwicklung eines Client-Server Spiels in Anlehnung an das Ravensburger Spiel „Memory“, wurde in der gegebenen Zeit erreicht. Die geleisteten Zeiten weichen jedoch teilweise von der abgegebenen Zeiteinschätzung ab.

Aufgabe	Zuständig	Stunden (Soll)	Stunden (Ist)	Anmerkung
Client-Implementierung	Fr. Klein Hr. Chakmehdouz	80 PS	87 PS	Die grafische Anzeige der Oberflächen funktionierte nicht sofort wie geplant.
Client-Testmanagement	Fr. Klein Hr. Chakmehdouz	25 PS	22 PS	
Server-Implementierung	Hr. Onutor	25 PS	28 PS	
Server & Kombi-Testmanagement	Hr. Onutor Hr. Chakmehdouz	40 PS	36 PS	Durch Hardwareschwierigkeiten mussten die Tests verschoben werden.
Dokumentation	Hr. Onutor	15 PS	13 PS	
Präsentation	Fr. Klein	13 PS	12 PS	
Gesamtsumme		**198 PS**	**198 PS**	

6.2 Projektverlauf und Fazit

Die größte Herausforderung des Projektes waren nicht die Planung oder Realisierung sondern das Zeitmanagement. Trotz der Erfahrungen in vorherigen Projekten im Betrieb und einer sorgsamen Planung war es nicht nur teilweise möglich die gegebenen Meilensteine am geplanten Tag einzuhalten. Eine Verzögerung der Projektabgabe ist dadurch jedoch nicht aufgetreten.

Die Realisierung des Projektes war für alle Beteiligten eine gute Erfahrung um die Tücken und Probleme in so einem IT-Projekt zu erleben und damit umzugehen.

Solche Probleme traten gerade bei den Personen auf für die die Programmiersprache Java etwas Neues war. Auch gab es hardwareseitig Schwierigkeiten gerade was die Zusammenarbeit und die Netzwerkverbindung angeht.

Abschließend ist zu sagen dass das Projektziel erreicht wurde und ein schönes und benutzbares Produkt entstanden ist welches dem Anwender Spaß bereiten sollte.