**Projektdokumentation der Semesteraufgabe „Skyjo als Multiplayer Spiel mit Python“**

Dies ist die Dokumentation der Entwicklung des Spiels Skyjo als Multiplayerspiel mit Python im Rahmen der Semesteraufgabe der Vorlesung Informatik II.

**Funktionen**

Unser Endprodukt besteht aus zwei ausführbaren Python Dateien, server.py und client.py. Um das Spiel zu starten, muss zunächst der Server (server.py) ausgeführt werden. Im Terminal werden dabei Statusmeldungen zu den laufenden Spielen angezeigt. Anschließend können sich beliebig viele Clients verbinden. Dabei geben sie ihren Spielernamen und den Namen des Spiels an, dem sie beitreten möchten. Wenn bei der Verbindung ein neues Spiel gestartet werden soll, muss zusätzlich angegeben werden auf wie viele Spieler der Server warten soll.

Technisch gesehen können auf dem Server beliebig viele Spiele gleichzeitig laufen, die maximale Spieleranzahl pro Spiel ist jedoch auf 4 Spieler begrenzt. (Im Code auf Github ist mit server.listen(10) die maximale Anzahl Clients insgesamt auf 10 begrenzt, dies kann aber einfach geändert werden).

Als grafische Benutzeroberfläche des Clients wird Pygame benutzt, auf dem Server läuft Python mit einigen anderen Modulen. Die Kommunikation zwischen Client und Server läuft über das socket Modul und das TCP-Protokoll, dies ist verbindungsorientiert und damit für ein Kartenspiel wie Skyjo besser geeignet als UDP, welches schneller, aber weniger zuverlässig ist.

**Zusammenfassung des Entwicklungsprozesses**

Um mit der Entwicklung zu beginnen, wurde zunächst ein Struktogramm erarbeitet, welches grob als Richtlinie für den Programmaufbau dienen sollte. Als Grundlage hierfür diente das bereits in der Aufgabenbeschreibung gegebene Struktogramm, welches während des Entwicklungsprozesses regelmäßig erweitert und konkretisiert wurde.

Nachdem nun klar war welche Unterprogramme notwendig sind, wurde eine Ordnerstruktur erstellt, wobei schon von Anfang an in die Zukunft geplant wurde, welche Klassen und Funktionen die Unterprogramme später mal enthalten sollten und wie diese miteinander in Zusammenhang stehen werden. Die Ordnerstruktur teilt die verschiedenen Unterprogramme auf in ein Modul des Servers, ein Modul des Clients und ein geteiltes Modul.

Für die weitere Entwicklung war es von Vorteil, das Gesamtziel in 3 Unterziele aufzuteilen: Um ein funktionierendes Endprogramm zu erhalten war es notwendig zuverlässig funktionierende Netzwerkfunktionen (Unterziel 1) zu programmieren. Danach kann man anfangen eine zuverlässige Spiellogik zu programmieren (Unterziel 2). Zuletzt kann man dann eine GUI erstellen mit der die Clients über die Netzwerkfunktionen Einfluss auf die Spiellogik auf dem Server nehmen können (Unterziel 3). Da diese Ziele alle aufeinander aufbauen erschien es als sinnvoll diese nacheinander in Angriff zu nehmen, was so auch umgesetzt wurde.

**Entwicklung der Netzwerkfunktionen**

Zu Beginn wurde ein neuer Branch erstellt, um die Netzwerkfunktionen zu entwickeln, dieser Branch, genannt „network – testing“ wurde benutzt, um die Netzwerkfunktionen nacheinander zu entwickeln und einzelne Funktionen zu testen bevor weitergemacht wurde. Zum testen dieser Funktionen wurde ein Raspberry Pi 5 als Server verwendet, was sich später als unnötig herausstellte, da Server und Client auch lokal auf demselben Gerät getestet werden konnten, indem man die Loopback-Adresse (Localhost bzw. 127.0.0.1) verwendete.

Die Netzwerkfunktionen, welche zu diesem Zeitpunkt nur ein Spiel gleichzeitig unterstützten wurden nach Fertigstellung und einigen Tests wieder in den development Branch gemerged. Da man sich im Team aber später so verständigte, dass man doch mehrere Spiele gleichzeitig laufen lassen wolle, wurde dann später im Branch „network\_multiple\_lobby-testing“ die Unterstützung für mehrere Spielrunden gleichzeitig hinzugefügt.

Damit waren die Netzwerkfunktionen nun nahezu fertig, wie sich später aber während des Entwickelns der Spiellogik herausstellte gab es hier noch ein Problem, um dies zu beheben wurden die Netzwerkfunktionen so umgebaut, dass bei jedem Senden von einem Datenstrom (Client zu Server und umgekehrt) im Header des Datenstroms die Länger der gesendeten Nutzdaten übermittelt wird, dadurch wissen beide Teilnehmer auf wie viele Bytes Nutzdaten genau gewartet werden muss, wodurch die gesendeten Pakete auch immer vollständig ankommen.

**Entwickeln der Spiellogik**

Zum Entwickeln und Testen der Spiellogik auf dem Server wurde wieder ein neuer Branch erstellt mit Namen „Server-Gamelogic“. Es wurden auch hier nacheinander Funktionen hinzugefügt, da diese aber schlecht getestet werden konnten ohne einen funktionierenden Client, musste relativ schnell ein Testclient umgesetzt werden. Dieser Client wurde als Kommandozeileninterface umgesetzt, bei dem der Nutzer über Befehle im Terminal Einfluss auf die Spiellogik nehmen kann, ändert sich der Zustand des Spiels geben dies die Testclients über print- Befehle aus. Ebenfalls werden im Terminal des Servers print- Befehle verwendet, die ein einfaches Debugging ermöglichten.

Es wurden hier nun nacheinander Funktionen hinzugefügt und Tests gemacht, bis ein erstes fertiges Spiel entstand. Nun konnte man bereits das Spiel spielen über die Terminals der Testclients. Daraufhin wurden zahlreiche Tests durchgeführt, um sicherzustellen, dass alle Aspekte der Spiellogik wie vorgesehen funktionierten. Nachdem dies sichergestellt war, wurden noch einige weitere Verbesserungen durchgeführt, wie zum Beispiel das Umstellen von einer Liste zu einem Dictionary, was einen effizienteren Zugriff auf die Inhalte ermöglicht.

**Entwicklung der GUI**

Nachdem das Spiel lauffähig war, wurde im letzten Entwicklungsschritt die grafische Benutzeroberfläche (GUI) umgesetzt, die den bisher verwendeten Testclient ersetzt.

Zur Funktion der GUI wird auf die Datei „Client Architektur Uebersicht.pdf“ im Dokumentationsordner auf Github verwiesen.

|  |  |
| --- | --- |
| **Aufgabe:** | **Umgesetzt von:** |
| Erstellen einer Ordnerstruktur | Jonas Heselschwerdt |
| Entwicklung von Struktogrammen, Aufstellen der 3 Unterziele (s.o.) sowie Erstellung eines Grundkonzepts | Jonas Heselschwerdt |
| Entwicklung und stetige Verbesserungen der Netzwerkfunktionen | Jonas Heselschwerdt |
| Entwicklung der Spiellogik auf dem Server | Jonas Heselschwerdt |
| Erstellen eines Testclients zu Debuggingzwecken | Jonas Heselschwerdt |
| Entwicklung der GUI | Ellijah Brauch |

**So kann das Spiel gespielt werden:**

**Start des Servers**: Wenn sich Client und Server im selben lokalen Netzwerk befinden, kann einfach server.py gestartet werden. An der Stelle server.bind(('0.0.0.0', 65111)) im Programmcode kann der Port, auf dem der Server läuft, geändert werden, falls nötig. Auf dem Server muss sonst nichts mehr gemacht werden während des Spiels.

**Bedienung des Clients**: Die Python Datei client.py muss nur gestartet werden, danach werden alle Eingaben wie Spielernamen, Spieleranzahl, Name der Spielrunde und die IPv4 des Servers abgefragt. Danach kann der Nutzer über die GUI das Spiel spielen.

**Bedienung des Testclients**: Zunächst muss in der Datei client\_CLI\_test.py in Zeile 17 angegeben werden wie viele Spieler in einer Spielrunde sein sollen (nur relevant, wenn mit dem Testclient das Spiel gestartet wird). Danach muss in Zeile 18 die IPv4 des Servers angegeben werden. Nun kann client\_CLI\_test.py gestartet werden. Es wird in der Konsole der Name des Spielers und der Name des Spiels abgefragt.

Nun kann mit den folgenden Befehlen Einfluss auf die Spiellogik genommen werden:

* flip x y (Um eine Karte an der Stelle x y aufzudecken, falls der Nachziehstapel aufgedeckt wurde)
* take from discard x y (Um die Karte vom Ablagestapel mit der Karte an der Stelle x y zu tauschen)
* take from draw x y (Um die Karte vom Nachziehstapel mit der Karte an der Stelle x y zu tauschen, der Nachziehstapel muss hierfür aufgedeckt worden sein)
* check draw (Um den Nachziehstapel aufzudecken)
* leave (Um das Spiel zu verlassen, client\_CLI\_test.py wird beendet)
* end (Um das Spiel zu beenden, dies ist nur möglich, wenn man es auch gestartet hat)

Hinweise zum Testclient: Der Testclient wurde zum Debuggen verwendet, deshalb werden auch Informationen angezeigt, die der Spieler teilweise nicht haben sollte. Zudem gibt es im Testclient einzelne Funktionen, die nicht im fertigen Client umgesetzt sind (siehe leave- und end- Befehl beim Testclient). In einer Spielrunde können sich grundsätzlich Testclients und normale Clients befinden.

**„Kartenkoordinaten“** x y der Befehle sind so definiert:

|  |  |  |  |
| --- | --- | --- | --- |
| Karte x=0 y=0 | Karte x=0 y=1 | Karte x=0 y=2 | Karte x=0 y=3 |
| Karte x=1 y=0 | Karte x=1 y=1 | Karte x=1 y=2 | Karte x=1 y=3 |
| Karte x=2 y=0 | Karte x=2 y=1 | Karte x=2 y=2 | Karte x=2 y=3 |

Die x und y-Achse sind somit im Vergleich zu mathematischen 2D-Koordinatensystemen vertauscht.

**Verwendung von künstlicher Intelligenz**

Der Transparenz wegen soll hier am Schluss darauf verwiesen werden, dass beim Programmieren Sprachmodelle wie ChatGPT 4o oder Copilot unterstützend zum Einsatz kamen.