Abschlussprojekt
Advanced Software-Engineering
Dokumentation

Blokus Project

Teilnehmer in dem Projekt sind

Kevin Wagner / Kai Pistol / Luis Eckert /
Cynthia Winkler

Betreuende Dozenten
Herr Bentle & Herr Kessler

Zusammenfassung

In der folgenden Dokumentation wird die Implementierung des beliebten Brettspiels "Blokus" in eine digitale Form behandelt. Ziel des Spiels ist es, strategisch Formen auf einem Spielbrett zu platzieren, um so viele Flächen wie möglich zu bedecken und gleichzeitig die Lege-Möglichkeiten der Gegner einzuschränken. Das folgende Dokument enthält detaillierte Beschreibungen zu den Projektrollen, dem Vorgehen, der Architektur, dem Softwaredesign und den genutzten Testmethoden.

Inhaltsverzeichnis

1 Einführung

1. **Anforderung**

   Das Projekt "Blokusïst ein Multiplayer-Netzwerkspiel, das auf einer Client-Server-Architektur basiert. Das Ziel des Spiels ist es, möglichst viele der eigenen Spielsteine auf dem Spielbrett zu platzieren. Dabei müssen die Steine eines Spielers, die dieselbe Farbe haben, ausschließlich an den Ecken miteinander in Berührung kommen, nicht an den Seiten.

   Eine wichtige Komponente des Spiels ist der Multiplayer-Aspekt. Spieler treten in einer Spiellobby, die auch einen Chat beinhaltet, gegeneinander an. Während des Spiels können die Spieler über einen In-Game-Chat mit Mitspielern kommunizieren. Falls nicht genügend menschliche Spieler zur Verfügung stehen, werden die offenen Plätze mit KI-Spielern aufgefüllt.

   Zu den Hauptmerkmalen des Spiels gehört die Implementierung des gesamten Regelwerks von "Blokus". Die Unterstützung für menschliche Spieler umfasst die Verwaltung von Benutzerkonten. Ein besonderer Fokus liegt auf der Entwicklung einer Künstlichen Intelligenz für das Spiel. Ziel ist es, mindestens eine KI zu schaffen, die bessere Züge als zufällige Bewegungen ausführt und somit eine Herausforderung für die Spieler darstellt.

   Das Spiel wird ausschließlich als Netzwerkspiel konzipiert, auch wenn es sich um Einzelspielerpartien handelt. Dabei werden offene Spielerplätze mit KI-Spielern besetzt. Es ist ausreichend, einzelne Spielrunden zu realisieren, ohne dass eine detaillierte Statistikführung oder Punktezählung strikt erforderlich ist.

2. **Ziel**

   Das Ziel unseres Projekts ist es, eine funktionstüchtige Anwendung zu entwickeln, die sich strikt nach den Vorgaben des Lastenhefts richtet. Dabei liegt ein besonderer Fokus darauf, die in der Vorlesung Ädvanced Software Engineeringërlernten Inhalte praktisch umzusetzen. Dies beinhaltet die Anwendung fortgeschrittener Softwareentwicklungstechniken und -prinzipien, die es uns ermöglichen, eine qualitativ hochwertige, effiziente und benutzerfreundliche Softwarelösung zu erstellen.

2 Anforderungen

Die Anforderungen für das Projekt wurden als Issuses in einem GitHub-Projekt angelegt. Hierbei wurde sich zuerst an den Vorgaben des zur Verfügung stehenden Lastenheftes orientiert und aus den " ´UseCases“ funktionelle Anforderungen abgeleitet. Weitergehend haben wir selbst nicht-funktionale Anforderungen entwickelt und diese ebenfalls als Issues formuliert. Für eine bessere Übersicht und Ordnung haben wir eigene Labels erstellt und zu den entsprechenden Anforderungen zugeordnet.
Mit der fortschreitenden Arbeit an dem Projekt haben wir notwendige Ergänzungen gemacht, sollte es zu folgenden Anforderungen kommen.

2.1 Funktionale Anforderungen

1. **Spielstart durch den Host:** Nur der Lobby-Host kann ein neues Spiel starten, wenn alle anderen Spieler bereit sind.
2. **Aktualisierung von Spielerprofilen:** Spieler können ihre Profile aktualisieren.
3. **Benutzernamen und Passwort festlegen:** Spieler können einen Benutzernamen und ein Passwort festlegen.
4. **Passwort zurücksetzen:** Spieler können ihr Passwort zurücksetzen.
5. **Platzieren von Steinen:** Spieler können im Verlauf des Spiels Steine auf dem Spielbrett platzieren.
6. **Ergänzung durch KI:** Wenn die maximale Spieleranzahl nicht erreicht ist, werden fehlende Spieler durch KI ersetzt.
7. **Spiel verlassen:** Spieler können ihr aktuelles Spiel verlassen.
8. **Chat im Spiel:** Spieler können anderen Spielern im Spiel Nachrichten senden.
9. **Lobby verlassen:** Spieler können eine Lobby verlassen.
10. **Spielende nach einer Runde:** Das Spiel endet, nachdem eine Runde gespielt wurde.
11. **Rückkehr in die Lobby nach Spielende:** Spieler kehren in die Lobby zurück, wenn ein Spiel endet.

2.2 Nichtfunktionale Anforderungen

1. **Visuelles Feedback bei Spieleraktivität:** Anzeige mit Farben (grün/gelb/rot), um die Aktivität des Spielers darzustellen.
    - Grün: Der Spieler ist aktiv, entweder durch Mausbewegungen oder weil das Spiel als aktiver Tab geöffnet ist.
    - Gelb: Der Spieler ist seit weniger als 5 Minuten inaktiv.
    - Rot: Der Spieler ist seit mehr als 5 Minuten inaktiv.

2. **Frustrationsfreie Spielerfahrung:** Gewährleistung einer angenehmen und stressfreien Spielerfahrung.

3. **Soundeffekte und Musik im Spiel:** Integration von Soundeffekten und musikalischer Untermalung im Spiel.

4. **Präferierte Spielerfarbe:** Ermöglichen der Auswahl einer bevorzugten Farbe durch den Spieler.

5. **Nutzernamen ändern:** Möglichkeit für Spieler, ihren Nutzernamen zu ändern.

6. **Verschiedene KI-Schwierigkeitsgrade:** Anbieten verschiedener Schwierigkeitsstufen für die KI, um unterschiedlichen Spielerfähigkeiten gerecht zu werden.

7. **Spielerbewertungen und Feedback:** Ein System, um nach dem Spiel Feedback zu Spielern oder dem Spiel selbst abgeben zu können.

8. **Mehrsprachige Unterstützung:** Unterstützung verschiedener Sprachen, um eine breitere Spielerbasis anzusprechen.

9. **Anzeigen eines Scorebords:** Nach dem Ende eines Spiels werden die Spieler auf ein Scoreboard weitergeleitet, um das Ranking zu präsenitieren

2.3 Definition of Done

1. **Code-Erstellung und Kommentierung:** Der Code für das Feature ist vollständig geschrieben und sorgfältig kommentiert, um Verständlichkeit und Wartbarkeit zu gewährleisten.

2. **Peer-Review des Codes:** Der Code wurde mit anderen Entwicklern besprochen und einer gründlichen Kontrolle unterzogen. Dies gewährleistet, dass er den Teamstandards entspricht und potenzielle Fehler oder Verbesserungsmöglichkeiten identifiziert werden.

3. **Erfolgreiche Unit-Tests:** Alle Unit-Tests für das Feature sind erfolgreich durchgeführt worden. Dies stellt sicher, dass der Code wie erwartet funktioniert und keine bekannten Fehler aufweist.

4. **Anfertigung einer vorläufigen Dokumentation:** Eine vorläufige Dokumentation des Features wurde in Stichpunkten angelegt. Dies dient als Grundlage für die spätere Erstellung einer vollständigen Dokumentation.

5. **Gruppenabnahme des Features:** Das Feature wurde von der gesamten Gruppe präsentiert. Diese kollektive Zustimmung stellt sicher, dass das Feature den Anforderungen und Erwartungen des Teams entspricht.

3 Vorgehen

Zu Beginn des Projektes haben wir ein GitHub-Reposistory aufgesetzt. In diesem haben wir Anforderungen, Aufgaben sowie den Code und die Dokumentation selbst verwaltet. Für die Dokumentation haben wir ein Overleaf-Projekt aufgesetzt. Und zum programmieren hat jeder eine für sich komfortable IDE genutzt, wie zum Beispiel PyCharm oder Visual Studio.
Wie bereits beschrieben wurden die Anforderungen als Issues in GitHub festgehalten. Für die konkreten Aufgaben haben wir ein GitHub-Projekt angelegt. In diesem konnten wir einzelne Aufgaben festlegen und diese sowohl Issues als auch Personen zuordnen. Die Aufgaben haben wir in folgende Kategorien unterteilt:

- No Status:
  Beinhaltet Aufgaben, die wir nicht direkt zuordnen können, die noch nicht angefangen wurden, aber im späteren Verlauf noch zu machen sind.
- Backlog:
  Beinhaltet den Aufgabenpool. Alle neu definierten Aufgaben kommen hier rein, um später in die Spalte Current Sprint oder In Progress verschoben zu werden.
- Current Sprint:
  Beinhaltet Aufgaben, die gerade anstehen und bis zum nächsten Meeting bearbeitet werden sollen, aber noch nicht angefangen wurden.
- In Progress:
  Beinhaltet Aufgaben, die gerade bearbeitet werden, die aus dem Current Sprint hier liegen, bis sie fertig sind.
- Done:
  Beinhaltet alle abgeschlossenen Aufgaben.

Zu Beginn des Projektes haben wir Rollen in unserem Team verteilt, siehe [nächstes Kapitel]. Folgend haben wir erste Aufgaben verteilt. Für diese haben wir zunächst keine festen Deadlines gesetzt, sondern nur darauf geachtet, dass in den folgenden Wochen daran gearbeitet wurde.

4 Rollen im Team

In einem der ersten Meetings haben wir im Team Rollen und grobe Aufgaben verteilt. Im Laufe des Projekts haben sich dieser erweitert und verändert. Folgende Auflistung bezieht sich auf den Stand bei Beendigung des Projekts:

- Kevin Wagner als Scrum Master und Entwickler
  Die Hauptaufgaben bestanden darin einen Überblick über des Projektes zu haben und somit zu planen wann welche Aufgaben erledigt sein sollen und die restlichen Teammitglieder daran zu erinnern ihre Aufgaben zu erledigen, sowie die Planung der Meetings. Zum Entwicklungsteil hat er eine Containerumgebung mit Docker aufgesetzt, die Basis der GUI (PyQt6) [ohne Netzwerkintegration] entwickelt und andere Mitglieder des Projektes von Zeit zu Zeit bei Fragen oder Problemen unterstützt.

- Luis Eckert als Product Owner und Entwickler
  Die Hauptaufgabe lag in der Planung der Aufgaben. Er hat die Aufgabenverteilung vorgenommen und überwacht. Als Entwicklungsteil hat er das Netzwerk aufgesetzt, so dass wir einen Server und Client haben, welche auch Multithreading können und richtig mit der GUI kommunizieren.

- Cynthia Winkler als Entwicklerin
  Die Hauptaufgabe lag in der Entwicklung der Spiellogik. Hierzu gehörte die Entwicklung eines funktionierenden Prototyps, welcher nach und nach mit weiteren Funktionen ausgestattet wurde, bis die Spiellogik komplett implementiert war. Als weitere Aufgaben gehörten noch das Identifizieren der Prozesse und die Entwicklung eines taktischen Mockups.

- Kai Pistol als Entwickler und Dokumentar
  Die Hauptaufgabe lag in der Gestaltung der GUI, wobei vom taktischen Mockup ausgegangen wurde, welches dann in Code übertragen wurde. Darüber hinaus wurde der AI- Player konzeptioniert. Ebenfalls wurde die Dokumentation, sowie das Tracking der offenen Punkte übernommen.

5 Architektur

Für die Architektur der Anwendung wurde die Django API als Grundlage für die Persistierung und Authentifizierung der Benutzerdaten ausgewählt. Diese API überträgt die relevanten Benutzerdaten zuverlässig an eine PostgreSQL-Datenbank. Dabei übernimmt sie nicht nur die Aufgabe der Datenpersistierung, sondern fungiert auch als Authentifizierungsschnittstelle zum Game-Service.
Der Game-Service selbst basiert auf einem leistungsstarken TCP-Socket, der die Möglichkeit bietet, sich gleichzeitig mit mehreren Benutzern über Multi-Threading zu verbinden. Diese Architektur ermöglicht eine effiziente und reibungslose Kommunikation zwischen den Benutzern und dem Game-Service.
Im Hinblick auf die Benutzeroberfläche des Spiels wurde ursprünglich auf die Implementierung von PY Simple GUI, TK-Inter und Py-Game gesetzt. Nach einer eingehenden Bewertung hinsichtlich Kompatibilität, Funktionalität und der Prämisse, einen überschaubaren Tech-Stack zu verwenden, wurde jedoch dieser Ansatz verworfen. Stattdessen entscheiden wir uns für PyQt6 als einheitliches UI-Framework. Diese Wahl ermöglicht nicht nur eine ansprechende und benutzerfreundliche Game-UI, sondern gewährleistet auch eine bessere Wartbarkeit des Systems und bessere Kompatibilität.

1. **Präsentationsschicht (Presentation Layer):**
   - Die Verwendung von "PyQt6"für die Game-UI gehört zur Präsentationsschicht.
   - Diese umfasst die Benutzeroberfläche und die Interaktionselemente der Anwendung.
2. **Anwendungsschicht (Application Layer):**
   - Die Nutzung der Rest API von Django zur Persistierung und Authentifizierung der Benutzerdaten fällt in die Anwendungsschicht.
   - Hier liegt auch die Zuständigkeit für die Koordination von Benutzeranforderungen und die Implementierung der Spiellogik.
3. **Datenzugriffsschicht (Data Access Layer):**
   - Die Übertragung der Benutzerdaten an eine Postgres-Datenbank durch die Django API gehört zur Datenzugriffsschicht.
   - Diese umfasst Datenbankabfragen, -einträge und -verbindungen und alle Operationen im Zusammenhang mit der Datenpersistierung.
4. **Infrastrukturschicht (Infrastructure Layer):**
   - Die Verwendung von TCP-Sockets im Game-Service gehört zur Infrastrukturschicht.
   - Hier gehören die Überlegungen zur Skalierbarkeit, Sicherheit und allgemeine Systemkonfiguration hin.
5. **Integrationsschicht (Integration Layer):**
   - Der Game-Service selbst, der auf einem TCP-Socket basiert und sich mit mehreren Benutzern verbinden kann, wird als Teil der Integrationsschicht betrachtet.

- Diese Schicht kümmert sich um die nahtlose Kommunikation zwischen verschiedenen Teilen der Anwendung.

Für das Projekt wird eine Client-Server-Architektur mit einem zentralen Server eingesetzt. Diese Architektur teilt sich in zwei Hauptkomponenten auf: den Client und den Server.

5.1 Client

Der Client in dieser Architektur ist eine Desktop-Anwendung, die als Schnittstelle zum Benutzer oder Spieler dient. Diese Anwendung ist in mehrere logische Teilbereiche gegliedert:

1. **Application**: Dieser Bereich befasst sich mit der Anwendung und allem, was nicht unmittelbar zum Spiel selbst gehört. Hierzu zählen Funktionen wie Registrierung, Login, Lobbies und Profiländerungen. Für die Umsetzung wird Pyqt6 verwendet.

2. **Game**: Der Game-Bereich konzentriert sich ausschließlich auf das Spielgeschehen. Auch hier kommt Pyqt6 zum Einsatz.

3. **Network**: Dieser Teilbereich dient als Bindeglied zwischen dem Server und dem Client. Für die Netzwerkkommunikation werden TCP-Sockets in Kombination mit Threading und Pyqt6 Signals verwendet.

5.2 Server

Der Server übernimmt die Verwaltung aller Spieler- und Benutzerinteraktionen, einschließlich der Verwaltung von Lobbies und laufenden Spielen. Er besteht aus zwei separaten Diensten:

1. **Authentication-Service**: Dieser Dienst ist für die Authentifizierung von Benutzern sowie für die Interaktion mit der Datenbank verantwortlich. Zur Implementierung wird die Django-REST API genutzt.

2. **Game-Service**: Der Game-Service verarbeitet Anfragen von Benutzern und Spielern. Er fungiert als zentrale Anlaufstelle für alle Serveranfragen und nutzt hierfür TCP-Sockets in Verbindung mit Threading.

Insgesamt bildet diese Architektur ein robustes Fundament für das Projekt, indem sie eine klare Trennung zwischen den verschiedenen Funktionen und Diensten ermöglicht und dabei eine effiziente Kommunikation zwischen Client und Server gewährleistet.

5.3 Client-Server Kommunikation

Für die Client-Server-Kommunikation wird ein barebone TCP-Server eingesetzt. Maßgebend für diese Entscheidung war die bereits rudimentär gesammelte Erfahrung im Rahmen des Semesters.

5.3.1 Datenpakete:

Für den Austausch von Informationen wurde ein einfaches Protokoll definiert. Im Rahmen des Projektes wird ein Datenpaket als Netzwerkevent bezeichnet. Jedes Paket bzw. Netzwerkevent besteht dabei aus zwei Teilen, der EventID und den Eventdaten. Zusätzlich gibt es noch den Header, der jedoch nur unmittelbar beim Senden und Empfangen eingesetzt wird.

5.3.2 Header:

Der Header ist eine Konstruktion, die zum Zeitpunkt der Übertragung Meta-Informationen übermittelt. Der Header ist genau 3 Byte lang (Bigendian). Byte 1 und 2 geben die genaue Bytelänge der Eventdaten an. Die maximale Länge der beträgt also 256*256 = 65536 Bytes (mehr als ausreichend). Byte 3 repräsentiert eine Event-ID. Die feste Länge des Headers erleichtert das Auslesen von Informationen aus dem Buffer. Der Buffer ist dabei ein temporärer Speicher, in den vom Netzwerk eingehende Nachrichten als Bytestrom abgelegt werden. Er agiert nach dem FIFO-Prinzip. Durch die Kenntnis der Header-Länge können die Bytes des Headers genau ausgelesen werden, ohne zusätzliche Informationen von anderen, später eingegangenen Nachrichten mit auszulesen. Zum gleichen Zweck wird auch die exakte Bytelänge der Eventdaten berechnet und im Header eingebettet.

5.3.3 EventID:

Die EventID wird zur Steuerung des Programmablaufs verwendet. Sie lässt sich mit einer Telefonvorwahl vergleichen. Während die Vorwahl bestimmt, in welches Land oder welche Stadt verbunden wird, gibt die EventID an, wo innerhalb des Systems eine Nachricht geleitet werden soll.
Für Blokus wurde basierend auf den Anforderungen und den initialen Überlegungen eine Liste von Eventcodes (NetworkEvent) definiert. Die Codierung ist positiv numerisch und beginnt mit 1. Zu jedem Code wurde außerdem eine Bezeichnung definiert, um als Entwickler/Leser besser damit arbeiten zu können. Im Programmcode wurde dies mit Hilfe einer Enum-Klasse umgesetzt (NetworkEvent). Manche der Codes werden nur von Clients gesendet oder empfangen, während andere nur vom Server gesendet werden.

- LOBBY_JOIN = 1
- LOBBY_CREATE = 2
- LOBBY_LEAVE = 3
- LOBBY_READY = 4
- LOBBIES_GET = 5
- GAME_START = 6
- GAME_PLACE_PIECE = 7
- GAME_UPDATE = 8
- GAME_INVALID_PLACEMENT = 9
- GAME_CHANGE_ACTIVE_PLAYER = 10

- GAME_FINISH = 11
- MESSAGE = 12
- SYSMESSAGE = 13
- PICK_COLOR = 14
- LOGIN = 15
- LOGIN_SUCCESS = 16
- REGISTRATION = 17
- REGISTRATION_SUCCESS = 18
- PROFILE_READ = 19
- PROFILE_UPDATE = 20
- PROFILE_DELETE = 21
- LOBBY_UPDATE = 22
- PROFILE_UPDATE_PASSWORD = 23

5.3.4 Eventdaten:

Die Eventdaten sind die zu einem Event gehörenden Informationen. Diese können rein informativer Natur sein und lediglich persistiert werden oder das Verhalten der Anwendung beeinflussen. Die Eventdaten werden als JSON verarbeitet. Mit Hilfe des JSON-Frameworks lassen sich die Daten einfach in Byteform (beim Versenden) und zurück als JSON-Objekt (beim Empfangen) transformieren. Der Grund für die Verwendung von JSON als Datenformat und Framework zur Transformation ist die 'sichere' Serialisierung und Deserialisierung von Objekten zur Laufzeit. Serialisierung ist ein Verfahren, mit dem Objekte zur Laufzeit in Text/Byteform transformiert werden können.
Die Deserialisierung von JSON-Objekten ist rigide und bietet über dessen Struktur keine Angriffsfläche für Missbrauch. Das Ergebnis einer JSON-Deserialisierung wird immer ein JSON-Objekt sein. Anders als beispielsweise bei der Deserialisierung durch das Pickle-Framework. Dieses erlaubt es zur Laufzeit, beliebige Objekte zu konstruieren, deren Initialisierung zur Ausführung von schädlichem Code führen kann. Wenngleich das durch die Verwendung von JSON ausgeschlossen und das Format selbst sicher ist, können die enthaltenen Daten dennoch schädlich sein, wenn sie unbedacht von der Anwendung verarbeitet werden.

5.3.5 Protokoll-Ablauf:

Im Folgenden wird der Ablauf der Kommunikation aus den Perspektiven 'Senden' und 'Empfangen' beschrieben.

- Senden:

    1. Event-ID auswählen

2. Eventdaten in JSON-Format serialisieren
3. Eventdaten-JSON in Bytestrom transformieren
4. Header aus EventID und Bytelänge des Eventdaten-Bytestroms ermitteln
5. Header senden
6. Eventdaten senden

- Empfangen:

1. Header aus dem Buffer mit Hilfe der festen Header-Bytelänge auslesen
2. Header in EventID und Eventdaten-Bytelänge auftrennen
3. Eventdaten aus dem Buffer mit Hilfe der ermittelten Event-Bytelänge auslesen
4. Eventdaten-Bytestrom in JSON-Objekt deserialisieren
5. Weiterverarbeitung des Events

5.3.6 Serverseitige Verarbeitung

Die zentrale Klasse der Serverseitigen Netzwerkkommunikation ist die Server-Klasse. Sie akzeptiert eingehende Verbindungen und verwaltet die eingehenden Netzwerkereignisse der Clients. Für jede neue eingehende Client-Verbindung wird ein neuer Thread gestartet, in dem das Senden und Empfangen von Daten erfolgt.
Verbindungen zu Clients, die das definierte Kommunikationsprotokoll verletzen, werden terminiert.
Für die Verarbeitung der Netzwerkereignisse wird das Strategy-Pattern verwendet: Die Basisklasse 'ServerEventHandler' stellt die Methode 'handleEvent' bereit, die von allen Derivatklassen überschrieben werden muss. Die Namen der Unterklassen beginnen alle mit 'ServerEventHandler_'.
In der Server-Klasse wird über Key-Value-Paare (Dictionary in Python) jeder Event-ID genau ein Serverhandler zugewiesen. Serverhandler können mehrfach verwendet werden.
Bei einem eingehenden Netzwerkereignis wird dann der zur angegebenen Event-ID gehörende Handler ausgewählt, und dessen 'handleEvent'-Methode wird aufgerufen.

- Authentifizierung

  Bis auf die Netzwerkereignisse 'LOGIN' und 'REGISTRATION' muss bei jedem Netzwerkevent ein Auth-Token angegeben werden, ansonsten terminiert der Server die Verbindung. 'LOGIN' und 'REGISTRATION' sind zwar Ereignisse, die der TCP-Server verarbeitet, jedoch wird die tatsächliche Logik in der mit Django REST-API implementiert.

- Validierung

  Der Server überprüft bei der Verarbeitung von Ereignissen, ob die angegebenen Eventdaten valide sind, insofern das möglich ist. Zum Beispiel, ob ein Spieler in einer Lobby ist oder in einem Spiel.

- Umwandlung von Netzwerkereignissen in PyQt-Signale

  Jedes eingehende Netzwerkereignis wird in ein PyQt-Signal umgewandelt. Dieser Prozess ermöglicht es, Netzwerkereignisse, die beispielsweise von einem Server oder

anderen Netzwerkquellen stammen, nahtlos in die Signal- und Slot-Architektur von PyQt zu integrieren. Durch diese Umwandlung können Netzwerkaktivitäten direkt in die Event-Handling-Logik der Benutzeroberfläche einfließen.

- Verbindung der UI mit Signalen

  Die Benutzeroberfläche (UI) unseres Programms kann sich mit diesen Signalen verbinden bzw. Handler für sie registrieren. Dies bedeutet, dass die UI auf die umgewandelten Netzwerkereignisse reagieren kann, als wären sie reguläre UI-Ereignisse. Die Möglichkeit, Handler für spezifische Signale zu registrieren, verleiht unserem Programm eine hohe Flexibilität und ermöglicht eine dynamische Reaktion auf Netzwerkaktivitäten.

- Kontrolle des Programms durch Signale

  Ein wesentlicher Vorteil dieses Ansatzes besteht darin, dass die Signale das Verhalten des Programms steuern können. Sobald ein Signal empfangen wird, löst es spezifische Aktionen oder Reaktionen innerhalb der Benutzeroberfläche aus. Dies kann alles von der Aktualisierung eines UI-Elements bis hin zur Auslösung komplexerer Geschäftslogik sein.

- Vorteile dieser Architektur:

  - Reaktionsfähigkeit: Unsere Benutzeroberfläche bleibt reaktionsfähig und kann dynamisch auf Änderungen und Ereignisse reagieren, die über das Netzwerk kommen.
  - Modularität: Durch die Verwendung von Signalen und Slots können wir unsere Netzwerkkommunikation und UI-Logik sauber trennen, was die Wartung und Erweiterung des Codes erleichtert.
  - Erweiterbarkeit: Diese Struktur ermöglicht es uns, zukünftig weitere Netzwerkereignisse und Funktionen ohne größere Überarbeitungen des bestehenden Systems zu integrieren.

  Insgesamt bietet dieser Ansatz eine effiziente und flexible Methode, um Netzwerkereignisse nahtlos in das UI-Design und die Programmsteuerung zu integrieren, und trägt wesentlich zur Benutzerfreundlichkeit und Leistungsfähigkeit unserer Anwendung bei.

6 Design

6.1 Anforderungsanalyse

Die Anforderungsanalyse basierte auf dem Lastenheft, welches detaillierte Anforderungen und Ziele sowohl für das Frontend-Design als auch das Backend-Design enthielt. In Zusammenarbeit wurde die Analyse weiter verfeinert und konkretisiert. Später wurden die Ergebnisse in einem Github-Repository zur gemeinsamen Nutzung und Verfolgung der Änderungen festgehalten. Die genauen Anforderungen sind unter 2 zu finden.

6.2 Designen des Frontends

6.2.1 Prozessanalyse

Aus den Anforderungen wurden die Prozesse, also die Programmabläufe, abgeleitet. Diese wurden als abgewandelte Ereignisgesteuerte Prozesskette (EPK) dargestellt, um die Abläufe und deren Zusammenhänge zu visualisieren.

6.2.2 Taktisches Mockup

Ein taktisches Mockup wurde erstellt, um eine erste, minimalistische Darstellung der einzelnen UI-Frames zu bieten. Dies basierte auf den funktionalen Anforderungen und den Programmabläufen. In jedem UI-Frame wurde die Anordnung der erforderlichen Elemente wie Buttons, Textfelder und Labels dargestellt.

6.2.3 Klick-Prototyp (Funktionelles Mockup)

Es wurde ein Klick-Prototyp als erstes haptisches Feedback der Anwendung entwickelt, bevor die eigentliche Entwicklung begann. Dieser Prototyp war schnell umsetzbar, da kein Coding erforderlich war und erlaubte eine einfache Anpassung und Visualisierung des Konzeptes. Die Möglichkeiten waren jedoch beschränkt: Das Spiel selbst konnte nicht abgebildet werden, nur die Anwendung drumherum.

6.2.4 Feedback einholen

Der Prototyp wurde innerhalb der Gruppe vorgestellt, um Feedback zu sammeln. Dies half nicht nur dabei, mögliche Verbesserungen zu identifizieren, sondern trug auch zur weiteren Definition der Machbarkeit bei.

6.2.5 Machbarkeitsprüfung

Die Machbarkeitsprüfung war nur bedingt möglich, da das erforderliche Know-how fehlte. Daher wurde das "fail-fast"Prinzip mit Prototypen angewendet. Die GUI wurde in Äpplicationünd "Gameäufgeteilt, um getrennte Zuständigkeiten zu schaffen.

6.2.6 Implementierung

Zunächst wurde ein Application-Prototyp mit PySimpleGui und ein Game-Prototyp mit PyGame erstellt. Es stellte sich heraus, dass PySimpleGui den Nachteil hatte, dass Fenster weder geschachtelt noch ein- und wieder eingeblendet werden konnten, was zu hohem Frust bei der Interaktion führte. Es gab auch mögliche Probleme bei der Integration der

Prototypen. Daher wurde die GUI (Application + Game) mit einem Framework, PyQt6, neu implementiert, basierend auf dem taktischen Mockup. Die Integration der Server-Client-Kommunikation in die GUI wurde durchgeführt, wobei darauf geachtet wurde, dass GUI und Netzwerkkommunikation sich nicht gegenseitig blockierten. Dies wurde durch getrennte Threads, die Verwendung von Queues und PyqtSignals erreicht. Netzwerk-Events wurden in PyqtSignals übersetzt, und die UI registrierte Handler-Funktionen für die jeweils relevanten Events. Die Steuerung der UI erfolgte durch Userinteraktionen und Netzwerkereignisse.

6.2.7 Testen und Iterieren

Die Tests in der UI wurden manuell durchgeführt, da die GUI häufig überarbeitet wurde und das Schreiben und Anpassen von Tests einen enormen Mehraufwand bedeutet hätte. Bei den Tests wurde die GUI auf die Kriterien "Funktionalität", "Korrekte Darstellungünd Äpplication Flow"geprüft. Es wurde sichergestellt, dass das Programm die erforderlichen Funktionen bereitstellte, sich am taktischen Mockup orientierte und alle erforderlichen Funktionen und Informationen bereitstellte. Außerdem wurde darauf geachtet, dass keine Sackgassen in der UI existierten und eine intuitive Navigation innerhalb der Anwendung gewährleistet war.

7 Testmethoden im Projekt

Im Rahmen unseres Projekts haben wir verschiedene Testmethoden eingesetzt, um die Qualität und Effizienz des Codes sicherzustellen.

7.1 Manuelle Tests:

Wir haben manuelle Tests durchgeführt, um sicherzustellen, dass die Anwendung in verschiedenen Szenarien korrekt funktioniert, insbesondere hinsichtlich Benutzeroberfläche, Funktionalitäten, Client-Server Kommunikation und Interaktionen der Module untereinander.

7.2 Unittests:

Unittests wurden implementiert, um die einzelnen Komponenten des Codes automatisch zu überprüfen. Diese Tests gewährleisten die korrekte Funktionalität der einzelnen Module und helfen dabei, mögliche Fehler frühzeitig zu erkennen.

7.3 Code-Reviews:

Code-Reviews waren ein integraler Bestandteil unseres Prozesses. Wir haben den Code durch andere Teammitglieder überprüft, um mögliche Fehler zu entdecken, Coding-Standards zu überprüfen und den Wissensaustausch im Team zu fördern.

7.4 Design-Reviews:

Design-Reviews konzentrierten sich auf die Gesamtarchitektur und Struktur der Anwendung. Unser Ziel war es sicherzustellen, dass sie besten Praktiken entspricht, skalierbar ist und zukünftige Erweiterungen erleichtert.

7.5 Pair Programming:

Wir haben Pair Programming genutzt, um Code in Echtzeit von zwei Entwicklern zu erstellen. Diese Methode hat nicht nur die Codequalität verbessert, sondern auch einen kontinuierlichen Wissensaustausch sowie gemeinsame Problemlösung ermöglicht.
Die Kombination dieser Testmethoden ermöglichte uns eine umfassende Qualitätskontrolle im gesamten Entwicklungsprozess. Wir konnten potenzielle Fehler frühzeitig identifizieren und eine robuste Softwarelösung entwickeln.

8 Verwendete Bibliotheken und Module

8.1 Authentication-Services:

- Django Version 4.2.7
- environs Version 9.5.0
- psycopg2
- djangorestframework
- django-cors-headers
- python-dotenv

8.2 Game-Service:

- requests
- numpy

9 Continuous Integration (CI)

In unserem Team haben wir eine vielfältige und effiziente Struktur für die Aufgabenteilung und Kommunikation etabliert, um eine optimale Zusammenarbeit zu gewährleisten. Dabei haben wir verschiedene Kommunikationskanäle genutzt, um sicherzustellen, dass alle Teammitglieder stets auf dem Laufenden sind und effektiv zusammenarbeiten können.
Das Herzstück unserer Codeverwaltung bildete das Git-Repository. Es diente als zentraler Ort, an dem sämtlicher Code gespeichert, aktualisiert und gemeinsam bearbeitet wurde. Dies ermöglichte es uns, Änderungen nahtlos zu integrieren und den Entwicklungsprozess kontinuierlich zu verbessern.
Für die Verwaltung unserer Aufgaben und das Tracking von Issues nutzten wir das Git Project Tool. Dieses Tool war unerlässlich, um unsere Arbeitsabläufe zu organisieren, Aufgabenprioritäten festzulegen und den Überblick über den Fortschritt einzelner Features und Bugfixes zu behalten. Die visuelle Darstellung von Aufgaben und deren Status in Git Project trug wesentlich zur Klarheit und Transparenz innerhalb des Teams bei.
Ein wesentlicher Aspekt unserer Zusammenarbeit war die Kommunikation, die sowohl mündlich als auch schriftlich über Discord erfolgte. Discord bot uns eine flexible Plattform, auf der wir sowohl in Echtzeit kommunizieren als auch wichtige Informationen und Dokumente teilen konnten. Die Vielseitigkeit von Discord ermöglichte es uns, unabhängig von Standort und Zeitzone effektiv zu kommunizieren und kooperieren. Durch regelmäßige Meetings und Chat-Konversationen konnten wir nicht nur den Code gemeinsam entwickeln, sondern auch Aufgaben koordinieren, Fortschritte verfolgen und eine kontinuierliche Kommunikation im Team sicherstellen.
Diese Kombination aus Git-Repository, Git Project und Discord stellte eine robuste und flexible Arbeitsumgebung dar, in der jedes Teammitglied seinen Beitrag leisten und gleichzeitig den Überblick über das Gesamtprojekt behalten konnte. Diese strukturierte und zugleich agile Arbeitsweise war entscheidend für den Erfolg unseres Projekts.

10 Dokumentation Benutzersicht

Setup des Projekts unter Windows

1. **Schritt 1: Installation und Start von Docker Desktop**

   Zuallererst müssen Sie Docker Desktop auf Ihrem Windows-System installieren und starten. Docker Desktop ist eine Anwendung, die es Ihnen ermöglicht, Docker-Container auf Ihrem lokalen System zu betreiben. Falls Sie Docker Desktop noch nicht installiert haben, können Sie es von der offiziellen Docker-Website herunterladen und die Installationsanweisungen befolgen. Nach der Installation stellen Sie bitte sicher, dass Docker Desktop läuft, bevor Sie mit dem nächsten Schritt fortfahren.

2. **Schritt 2: Verwenden der Kommandozeile im Projektverzeichnis**

   Als Nächstes benötigen Sie die Kommandozeile (CMD), um bestimmte Befehle auszuführen. Navigieren Sie dazu in das Projektverzeichnis, speziell in den Unterordner /**Project**/**blokusserver**/. Dies können Sie tun, indem Sie den Explorer öffnen, zum entsprechenden Verzeichnis navigieren, die Adressleiste anklicken, `cmd` eingeben und Enter drücken. Alternativ können Sie auch die Kommandozeile öffnen und den `cd` Befehl verwenden, um in das Verzeichnis zu navigieren.

3. **Schritt 2.1: Ausführen von Docker Compose**

   Im CMD-Fenster, das sich nun im Verzeichnis /**Project**/**blokusserver**/ befindet, führen Sie den folgenden Befehl aus: `docker-compose up -d`. Dieser Befehl startet die notwendigen Docker-Container im Hintergrund (dank des `-d`-Flags für "detached"). Dieser Schritt setzt voraus, dass eine `docker-compose.yml`-Datei im Verzeichnis vorhanden ist, die definiert, welche Container gestartet werden sollen.

4. **Schritt 3: Start der Anwendung**

   Zum Schluss starten Sie die Anwendung, indem Sie in das Verzeichnis /**Project**/**client**/ wechseln und dort die Datei `Blokus.exe` ausführen. Dies können Sie entweder über den Explorer tun, indem Sie zum Verzeichnis navigieren und die Datei doppelklicken, oder über die Kommandozeile mit dem Befehl `start Blokus.exe`, wenn Sie sich bereits im richtigen Verzeichnis befinden. Beachten Sie, dass das Starten der Anwendung je nach Systemleistung und anderen Faktoren etwa 10 bis 20 Sekunden dauern kann.

11 Architektur der Anwendung

11.1 Imports

In dieser Architektur werden mehrere Module importiert, darunter grundlegende wie `sys`, `threading`, `typing`, `PyQt6`, sowie einige benutzerdefinierte Module wie `qt6networkadapter`, `menu`, `settings` und `network`. Diese Module liefern die notwendigen Funktionen und Klassen, um die Anwendung zu unterstützen.

11.2 BlokusUtility-Klasse

Diese Hauptklasse erbt von `QMainWindow` und repräsentiert das Hauptfenster der Anwendung. In dieser Klasse werden verschiedene Funktionen implementiert:

- `__init__`: initialisiert das Netzwerk, die Benutzeroberfläche, die Einstellungen und das Menü.
- `__on_sys_message`: ist verantwortlich für die Behandlung von Systemnachrichten, die vom Netzwerk empfangen werden.
- `showAlert`: dient dazu, Benachrichtigungen anzuzeigen.
- `closeEvent`: stoppt das Netzwerk und akzeptiert das Schließen des Fensters.
- `initUI`: initialisiert die Benutzeroberfläche des Hauptfensters.
- `_createMenuBar`: erstellt die Menüleiste des Hauptfensters.
- `settingsFrame`: ist für die Anzeige des Einstellungsfensters zuständig.

11.3 AlertWidget-Klasse

Diese Unterklasse von `QWidget` wird verwendet, um Benachrichtigungen anzuzeigen.

- `__init__`: initialisiert das Widget und definiert seine geometrische Anordnung.
- `showAlert`: zeigt eine Benachrichtigung an und startet einen Timer.
- `__on_timeout`: wird aufgerufen, wenn der Timer abläuft, und entfernt das Benachrichtigungs-Widget.

11.4 AlertBoxWidget-Klasse

Eine weitere Unterklasse von `QWidget`, die für die Anzeige einer einzelnen Benachrichtigung verwendet wird.

- `__init__`: initialisiert das Widget und definiert seine Größe und seinen Stil.

11.5 Hauptprogramm

Das Hauptprogramm nimmt eine zentrale Rolle in der Anwendungsarchitektur ein. Es initialisiert die `QApplication`, setzt die Schriftgröße, erstellt eine Instanz der `BlokusUtility`-Klasse und zeigt dieses Hauptfenster maximiert an. Die Anwendung wird beendet, wenn das Hauptfenster geschlossen wird.
Diese Architektur ermöglicht eine klare und strukturierte Darstellung der verschiedenen Komponenten und deren Funktionen innerhalb der Client-Anwendung.

11.6 Gamelogik

Die Gamelogik wurde getrennt von der restlichen Anwendung gebaut. Das Ziel hierbei war die Erstellung eines funktionierenden Spieles, welches allein über Befehle mit der Konsole gespielt werden kann. Das Vorgehen war wie folgt:

1. **Erstellung eines Spielfeldes:** Es musste ein Spielfeld erstellt werden. Dieses ist ein quadratisches, zweidimensionales Array. Es wird als Objekt initialisiert und bei der Initialisierung kann die Größe angegeben werden. Dies war notwendig für die Tests, damit bei diesen nicht immer mit einem 20x20 Feld gearbeitet werden muss.

2. **Erstellung eines Spielsteines:** Es musste ein Spielstein erstellt werden. Dies wurde mit dem Package *numy* realisiert. Jeder Spielstein war ein zweidimensionales Array. Dabei bestand es immer aus Nullen und Einsen. Die Nullen stellen leere Stellen im Array dar und die Einsen den Spielstein selbst.

3. **Platzierung des Spielsteines:** Der erstellte Spielstein muss platziert werden. Dies wurde über Startkoordinaten (x, y) gemacht. Hierzu wurden mehrere Funktionen geschrieben, welche das Platzieren und überprüfen des Teiles übernehmen. Es musste unterschiedene werden zwischen dem ersten Teil, das platziert werden soll und allen weiteren.

    - Das erste Teil muss immer in einer Ecke liegen. Hierzu waren andere Überprüfungen notwendig, da auf wendiger geachtet werden muss.
    - Bei allen weiteren Teilen musste überprüft werden, dass sie nicht neben einem Teil der gleichen Farbe liegen, aber ein Teil der gleichen Farbe mit einer Ecke berühren.

4. **Implementation der AI:** Für die AI war es notwendig, dass mit einem einfachen Befehl ein Teil platziert wird, welches alle Bedingen erfüllt. Hier war ebenfalls eine Unterscheidung zwischen dem ersten und allen weiteren Spielsteinen notwendig. Zusätzlich musste ermittelt werden, wo und wie ein weiteres Teil platziert werden kann. Hierzu wurden mehrere Attribute, die entweder temporäre Daten oder langfristige Daten speichern, angelegt. Nach jeder Platzierung eines Steines wurden mögliche Plätze auf dem Spielfeld ermittelt, diese wurden mit (x, y) Koordinaten abgespeichert, sowie die Angabe, um welche Ecke es sich handelt. Anhand dieser Angabe können die Startkoordinaten des neuen Teiles einfach berechnet werden.

12 Fazit

12.1 Positiv Aspekte des Projekts

- **Erweiterung von Kenntnissen:** Ein großer Erfolg des Projekts war die Möglichkeit, unsere Kenntnisse und Fähigkeiten zu erweitern. Insbesondere konnten wir unsere Kompetenzen im Bereich Software Engineering vertiefen. Diese Kenntniserweiterung war nicht nur für das aktuelle Projekt wertvoll, sondern wird auch für unsere zukünftige Projekte eine wichtige Rolle spielen.

- **Funktionierendes Grundspiel:** Ein weiterer wesentlicher Erfolg war, dass das grundlegende Spiel funktionstüchtig ist. Dies ist ein bedeutsamer Meilenstein, der zeigt, dass die Kernidee des Spiels realisierbar ist und das Potenzial hat, weiterentwickelt und verfeinert zu werden.

12.2 Herausforderungen und Verbesserungspotenzial

- **Vorherige Überlegungen zur Verbindung von UI, Logik und Netzwerk:** Eine Herausforderung war die Integration von Benutzeroberfläche, Spiellogik und Netzwerk. In zukünftigen Projekten würden wir von Beginn an klare Schnittstellen definieren, um das Zusammenführen dieser Komponenten zu vereinfachen. Eine bessere Planung im Vorfeld könnte die Entwicklungseffizienz erheblich steigern.

- **Frühzeitige Festlegung auf ein UI-Framework:** Wir haben erkannt, dass eine frühere und präzisere Entscheidung bezüglich des UI-Frameworks für das Spiel und das Menü hilfreich gewesen wäre. Eine klare Festlegung zu Beginn des Projekts hätte die Entwicklung beschleunigen und mögliche Inkonsistenzen vermeiden können.

- **Mehrspielerfunktionalität:** Obwohl das Spielen mit mehreren Spielern möglich ist, gab es hierbei einige Herausforderungen. Für zukünftige Projekte wäre es sinnvoll, den Mehrspieleraspekt früher und intensiver in die Entwicklung einzubeziehen.

- **Status der KI:** Die Entwicklung der KI steht momentan noch auf unsicheren Füßen. Dies ist ein Bereich, in dem wir in Zukunft mehr Ressourcen und Zeit investieren müssen, um eine besser funktionierende KI zu erstellen.

12.3 Ausblick

Für die Zukunft wäre es denkbar, dass wir das Spiel durch das Hinzufügen weiterer Funktionalitäten auszubauen und die KI zu verbessern. Ziel ist es, ein noch ansprechenderes und herausforderndes Spielerlebnis zu schaffen. Wir sehen Potenzial in unserer Arbeit und sind der Meinung, dass die weiteren Entwicklung und Optimierung des Spiels erfolgreich wäre.