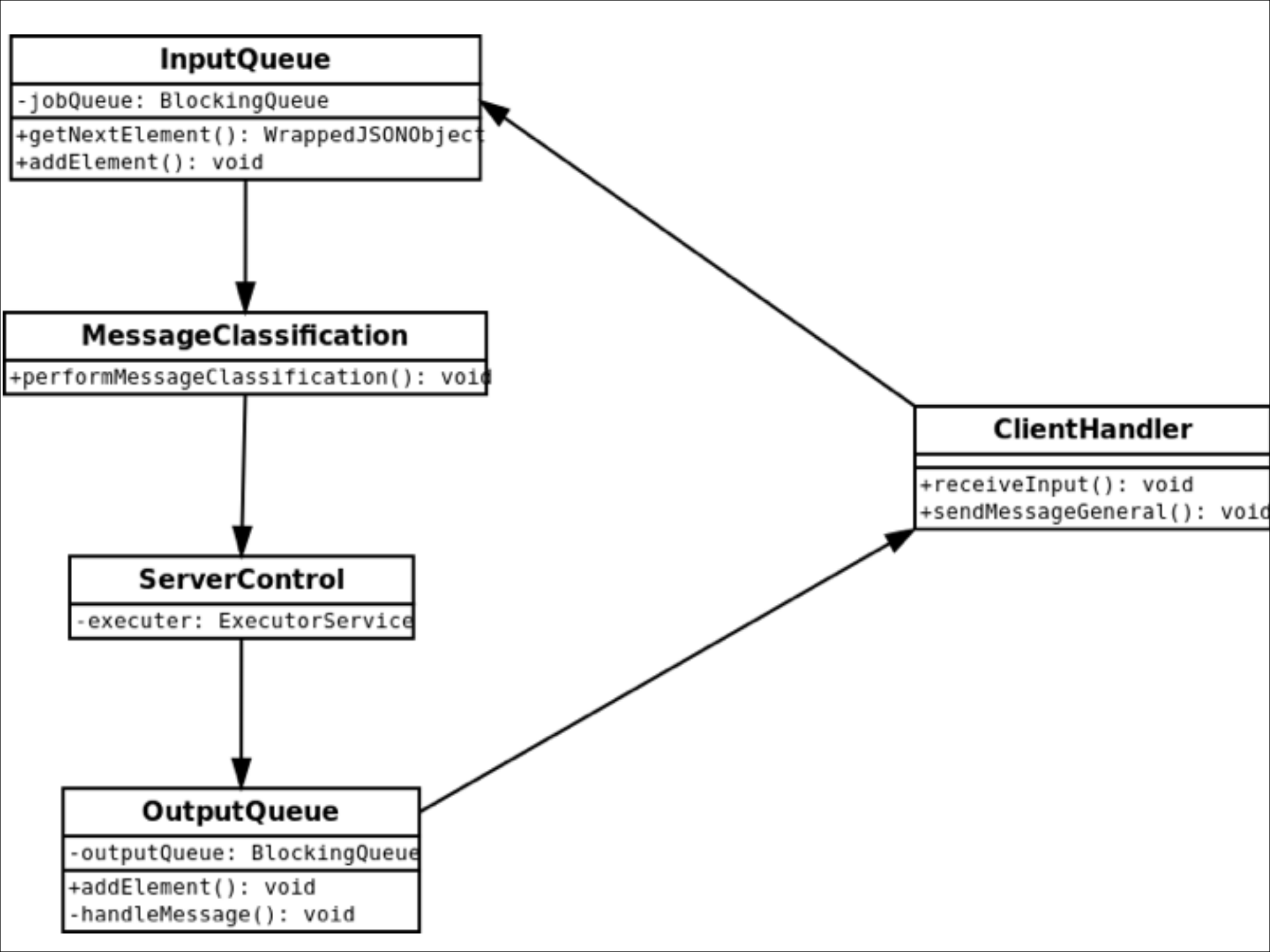
Message Abarbeitung Server

- Multi-threaded → ClientHandler
- Verwendung von zwei BlockingQueues
    - InputQueue
    - OutputQueue
- Zwei Executor, die die messages bearbeiten
    - Executor in ServerControl

      → Abarbeitung incoming messages
    - Executor in OutputQueue

      → Senden ausgehender messages


InputQueue
-jobQueue: BlockingQueue
+getNextElement(): WrappedJSONObject
+addElement(): void
MessageClassification
+performMessageClassification(): void
ServerControl
-executer: ExecutorService
OutputQueue
-outputQueue: BlockingQueue
+addElement(): void
-handleMessage(): void
ClientHandler
+receiveInput(): void
+sendMessageGeneral(): void

- Vorteile:
  - FIFO → sequentielle Abarbeitung der messages
  - Keine Race Conditions
  - Übersichtliche Struktur
  - Verwendung von Executorn

Kommunikation Observer

- Serverseitige Concurrent LinkedQueue
- Clientseitige Concurrent LinkedQueue

Protokollerweiterung / Extensions

- "x" : <int>, // x-Koordinate des Spielzugs
  "y" : <int>, // y-Koordinate des Spielzugs
  "rotation" : < 0 | 1 | 2 | 3 >, // Drehung des Plättchens
  "placement" : <int> // Gebietsindex fürr zu setzenden Meeple (optional)

- Wurde erweitert um :
- “specialMeeple" : <String> // EnumType (Bishop oder Bigmeeple) (optional)

- Server speichert bei Login Capabilities und zeigt dem Client nur Spiele mit passenden Extensions an

PlayerThread / Serverstabilität

- TileDrawn wird von einem extra Thread bearbeitet

- Eigener virtueller Server war eine große Hilfe
  -> Stabilitätsmaßnahmen
  1. Null Strings
  2. Unvollständige Messages
  3. Falsche Messages

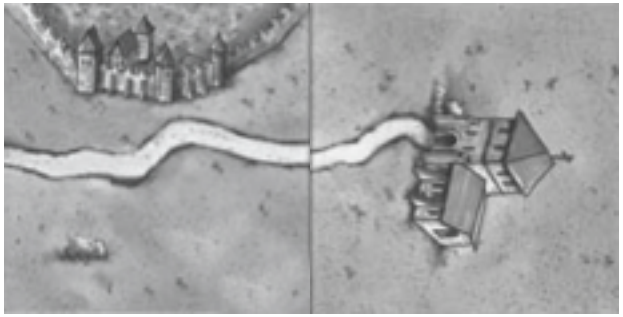
Areas: [Meadow, Road, Meadow, Town]

SingleAreas: [$Meadow_1$, $Road_1$, $Meadow_2$, Town]

$SingleAreasOnCard_1$: [M_1, R_1, M_2, T]

$SingleAreasOnCard_2$: [M_3 ,R_2, Cloister]

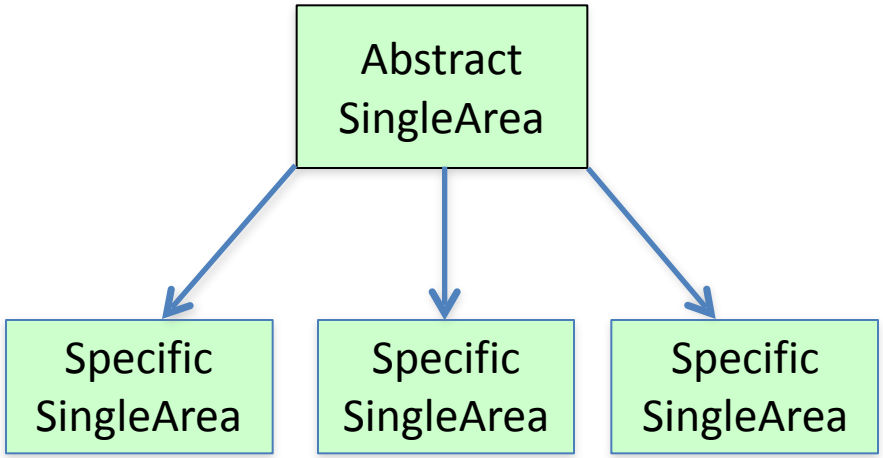
MergedSingleAreas: [M_1, R_1, T, Cloister]



- Logik in Klasse „ServerGame“, „Game“ und einzelnen Klassen, welche die verschiedenen SingleAreas repräsentieren.

- Ablauf
  - createInitialCardAndSingleAreas()
  - createSingleAreasForCard(...)
  - mergeEdges(...)

List<AbstractSingleArea> singleAreas
  - Kleinere Indizees der Liste entsprechen älterer SingleArea
  - Neuere Indizees werden beim Merge-Vorgang mit älteren überschrieben



Zugmöglichkeiten der AI

- Einführung der Klasse PossiblePlacement
    - Position
    - Rotation
    - Meeple-Placement
    - Score
    - Priority

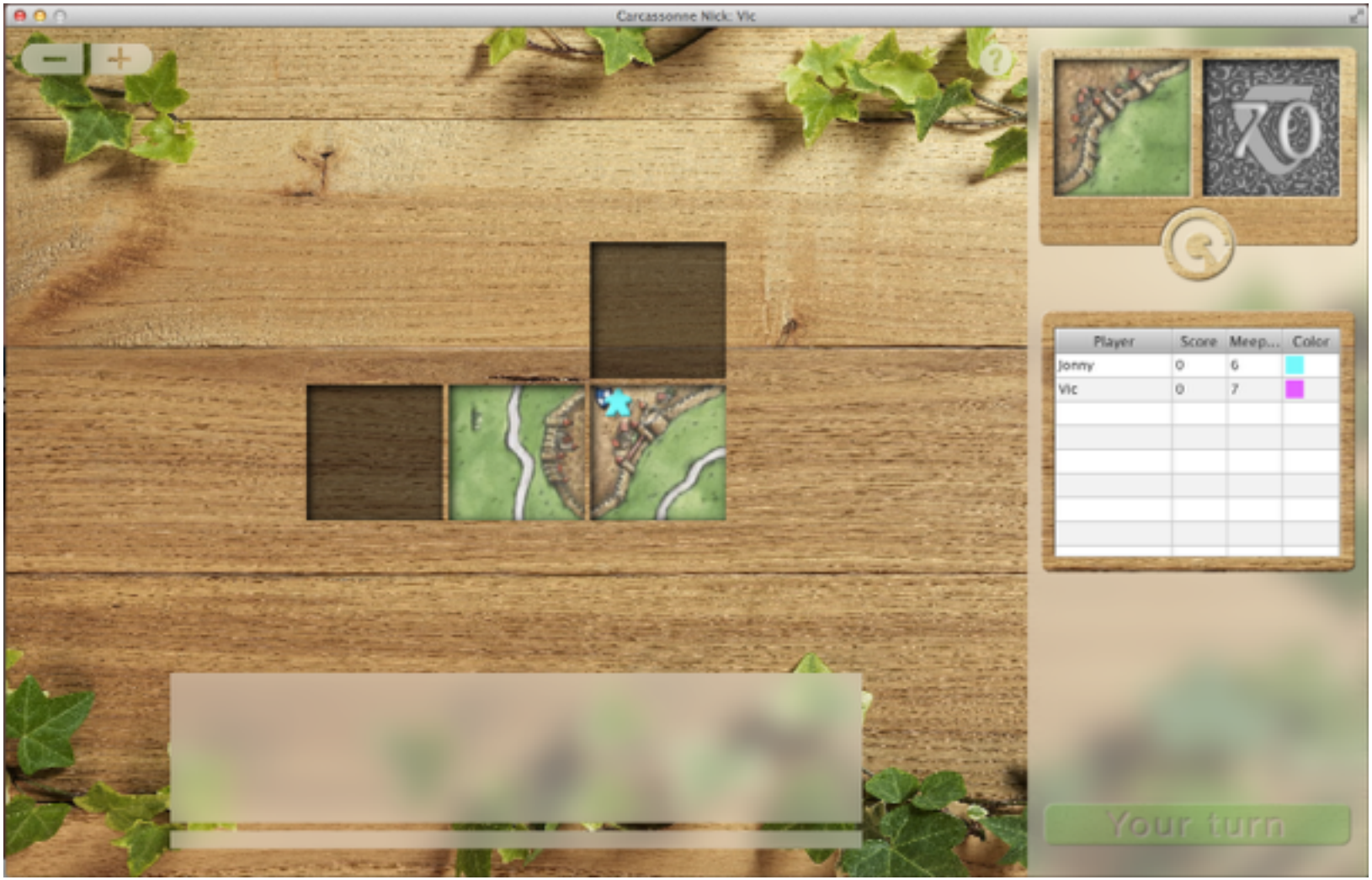
- Herausfinden der möglichen Züge
  - checkForLegalPlacements - Methode

- Herrausfinden der möglichen Meeple-Platzierungen
    - possibleMeeplePlacement - Methode

- Herausfinden des bestmöglichen Zuges
    - getBestPlacement - Methode
        - Überprüfung ob Gebiete abgeschlossen werden können
        - wenn nicht, werden die Prioritäten wie folgt gesetzt:
        - Kloster erweitern
        - Stadt erweitern
        - Straße erweitern

GameView

GameView


Carcassonne Nick: Vic
Player
Score
Meep...
Color
Jonny
0
6
Vic
0
7
Your turn

Big Meeple

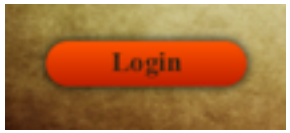
- Änderung in der Mehrheitsberechnung der Single Areas
- Abfrage ob Big Meeple schon gesetzt wurde

Bishop

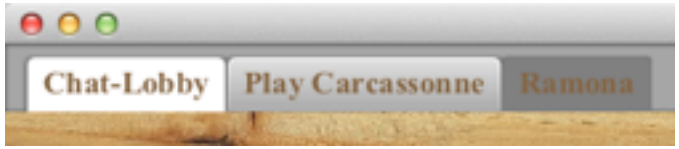
- Hinzufügen einer Mehrheitsberechnung für Bischöfe
- SingleAreaMeadow speichert adjazente Klöster in einer weiteren Liste analog zu Städten
- Endbewertung von Städten angepasst
- Schwierigkeit: Darf ich meinen Bischof setzen?

- DropDown Menü Farbe auswählen

- Für alle Buttons „DropShadow“



- Blinkender Tab beim Private Chat



- Grunddesign GameView, Login- und ChatLobby Fenster, ServerView

Card Design
(inkl. Erweiterungscards)

Piratendesign

Friedhofdesign

Einzelteile:

Meeple Design

Piratenmeeple

Bischofmeeple

Friedhofmeeple

Piraten-Bischofmeeple

Friedhof-Bischofmeeple

Big Meeple