

Dokumentation
Vier Gewinnt
GUI Anwendung mit dem QT Framework

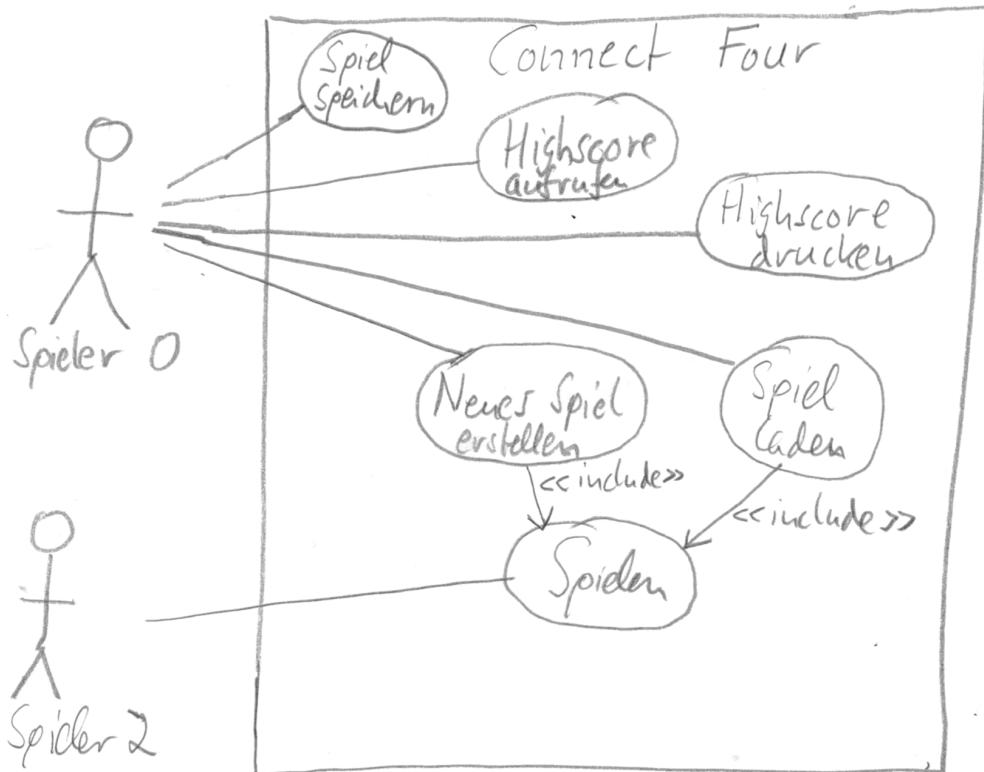
Beleg im Rahmen der Veranstaltung
Entwicklung von Multimediasystemen
Angewandte Informatik
HTW-Berlin

Emanuel Kern
s0535955@htw-berlin.de

1. Einleitung

Im Rahmen des Belegs wurde eine GUI Anwendung mit Hilfe des Frameworks QT erstellt, mit der sich das Gesellschaftsspiel Vier Gewinnt an einem PC spielen lässt. Dabei sollte die Möglichkeit bestehen gegen einen zweiten Spieler oder den PC anzutreten.

2. Use Cases



Spielen: Die Spieler können abwechselnd mit den Pfeiltasten und der Entertaste eine Reihe auswählen und einen Spielstein (Token) platzieren. Wird ein Spieler vom Computer gesteuert, so führt dieser seine Züge automatisch aus.

Spiel laden: Ein neues Spiel kann entweder durch das Laden oder „Neues Spiel erstellen“ begonnen werden. Beim Laden wird aus einer Datenbank ein Spiel ausgewählt (nicht über den Filebrowser).

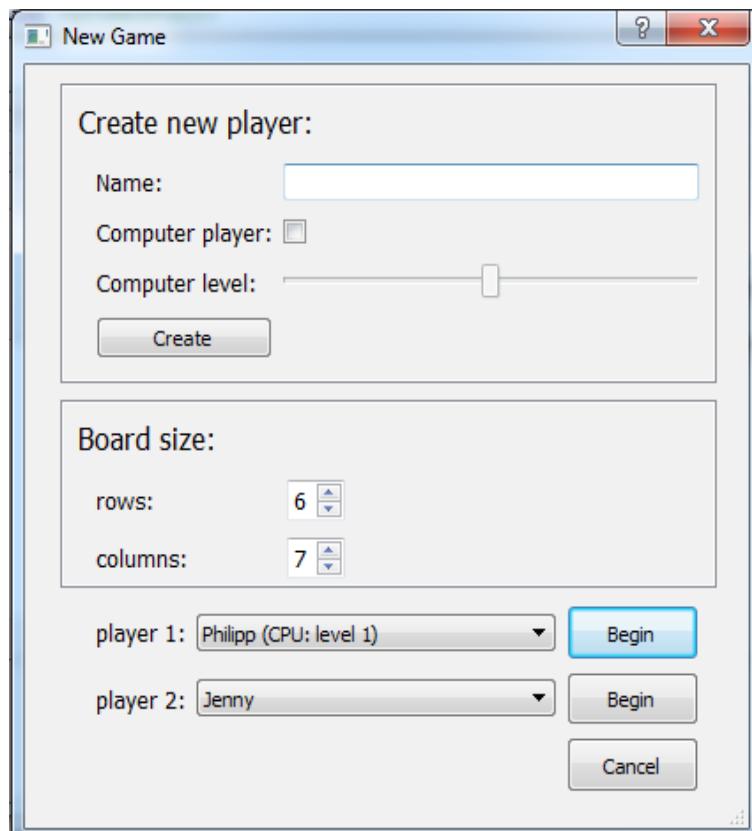
Spiel speichern. Ein laufendes Spiel wird in der Datenbank gesichert.

Highscore aufrufen: Eine Liste aller beendeten Spiele, die einer der Spieler gewonnen hat wird angezeigt. Die Liste ist nach der Anzahl der Züge sortiert, so dass die schnellsten Siege zu oberst liegen.

Highscore drucken: Druckt eine Ansicht der Highscore über einen standard Druckerdialog aus.

Neues Spiel erstellen: Ein neues Spiel kann mit diversen Optionen erstellt und dann gestartet werden. Dazu gehören.

- Wahl des Spielers (Name)
- Erstellen neuer Spieler
- Wahl des Startspielers
- Einstellen der Spielfeldgröße (4-9x 4-9)



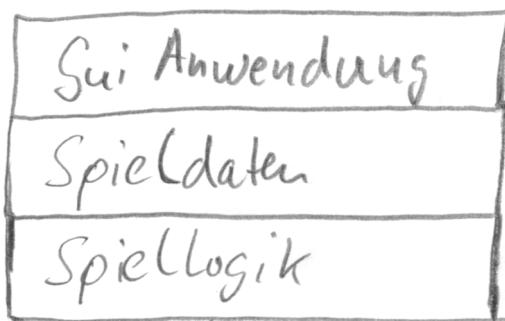
3. Architektur

Die Gesamtarchitektur des Projekts folgt keinem grundlegendem Paradigma oder Design Pattern. Insbesondere wurde das Model View Controller Pattern nicht eingehalten.

Trotzdem sind einige Merkmale der Gesamtarchitektur hervorzuheben:

3.1. Schichten

Alle Klassen lassen sich in 3 Schichten unterteilen, wobei jede höher liegende Schicht von den darunter liegenden abhängig ist. Im Umkehrschluss können niedrige Schichten benutzt werden um die Anwendung darauf basierend neu zu konzipieren.



Die Spiellogiksicht enthält die Klassen, die notwendig sind, um ein Vier Gewinnt Spiel rein theoretisch durchzuführen. Spieler haben keine Namen, sondern werden nur durch einen Index dargestellt und die Klassen sind nur dazu in der Lage ein einziges Spiel zu repräsentieren.

Die Spiellogiksicht basiert nicht auf QT-Klassen oder Aufrufen, sondern nur auf C++ Standardbibliotheken.

Die Spieldatenschicht erlaubt es Metadaten über mehrere Spiele zu persistieren, z.B. Spielernamen, Spielverläufe, Zeitpunkt eines Spiels. Sie stellt diese Daten aber nicht grafisch dar.

Die Spieldatenschicht greift auf nicht GUI Elemente von QT zurück.

Die GUI Anwendungsschicht vereint sowohl die grafische Repräsentation aller Daten als auch die Nutzerinteraktion um das Spiel zu steuern. Es gibt hier keine klare Trennung.

Die Klassen der GUI schicht erben von GUI Elementen des QT Frameworks.

Da in der GUI Anwendungsschicht, die Funktionalität und die grafische Darstellung in einer Schicht liegen, werden die zu einem GUI Objekt gehörenden Funktionen auch auf diesen Objekten direkt ausgeführt. Insbesondere hält die Hauptfensterklasse die Methoden, die mit den Use Cases der Anwendung korrespondieren.

4. Klassen

MainWindow	
GameView	
HighScoreTableView	
LoadGameTableView	
NewGameDialog	
GameScene	
PrimitivesGameScene	
SimpleGameScene	
SvgGameScene	
TileGraphicsItem	
TilePixmapGraphicsItem	
TileSvgGraphicsItem	
TilePrimitivesGraphicsItem	
TablePushButton	
TokenFallSound	
Daten	Same DataBase New Game Setup
Spieldaten	FourInARowGame FourInABoard RowField CPU Player

4.1 Klassen der Spiellogikschicht

FourInARowGame

- ermöglicht das Durchführen eines Spiels
- enthält Methoden zur Konfiguration des Spiels
- gibt nach jedem Zug den Status des Spiels zurück

FourInARowBoard

- Datenstruktur die den Zustand des Spielbretts vollständig repräsentiert
- direkter Zugriff zur Manipulation
- eine Instanz wird von FourInARowGame gehalten

RowField

- zusätzliche nicht vollständige Repräsentation des Spielfelds:
RowField stellt einen 3 dimensionalen Raum von möglichen 4er Reihen innerhalb des Spielfelds dar. Die Dimensionen ergeben sich aus den möglichen Startpunkten (Spalte, Reihe) und der Richtung. Wobei es 4 Richtungen gibt um alle Reihen eindeutig zu beschreiben. Willkürlich gewählt sind das: senkrecht nach oben, nach oben rechts, nach rechts und nach unten rechts vom ersten Spielstein der Reihe ausgehend.
An jedem Punkt dieses Raumes ist gespeichert, wie viele Spielsteine bereits in der Reihe liegen und wem diese gehören. Sobald zwei verschiedene Spieler die Reihe belegt haben ist die Anzahl nicht mehr relevant. Offensichtlich kann jeder gesetzte Spielstein mehreren potentiellen Reihen zugeordnet werden. Deswegen wird bei jedem Zug der erfolgreich ausgeführt wird, jeder Reihe, die dieser Stein angehören kann im RowField ein Spielstein hinzugefügt und direkt geprüft, ob die Anzahl 4 erreicht hat. Auf diese Weise ist nach jedem Zug sehr schnell zu ermitteln, ob das Spiel beendet wurde.
- Jede Position im RowField nimmt nur ein Byte in Anspruch. Die niedrigeren 4 bit repräsentieren die Anzahl, die höherwertigen 4 den Besitzer (kein, spieler0, spieler 1, beide)

CPUPlayer

- Objekte dieser Klasse können eine Referenz auf ein FourInARowGame auswerten und darauf basierend einen Zug für das Spiel auswählen. Der Zug wird nicht vom CPUPlayer selbst ausgeführt, sondern nur ein Int zurückgegeben.

4.2 Klassen der Datenschicht

NewGameSetup

- Datenstruktur die alle Daten kapselt, die den Status eines Vier gewinnt Spiels beschreiben, inklusive Metadaten, wie Spielernamen, Spielertyp, Schwierigkeitsgrad der Computerspieler.
- Dient als Schnittstelle zwischen der GUI und der Spiellogik.

Game DataBase

- SQLitedatenbank schnittstelle
- Stellt Methode zur Verfügung um QSqlQueries und QSqlQueryModels zu erhalten, die die Informationen enthalten, die von der GUI dargestellt werden sollen
- stellt Methoden zur Verfügung um Spiele zu speichern oder aus den gespeicherten Daten neu zu laden

4.3 Klassen der GUI Anwendungsschicht

MainWindow

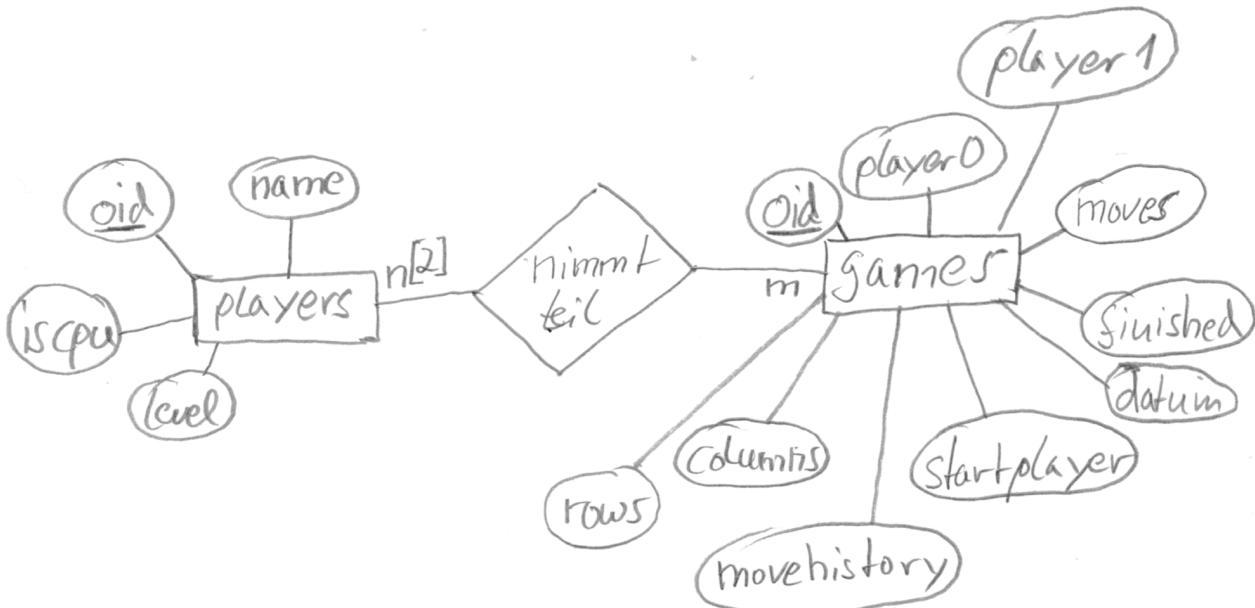
- erweitert QMainWindow und stellt das komplette Fenster der Anwendung dargestellt
- bietet Slots für die Use Cases an
- Schnittstelle für Interaktionen der verschiedenen Komponenten
- hält Referenzen (direkte/indirekte) Referenzen zu den anderen GUI Elementen

GameView

- erweitert QGraphicsView und bietet die Zeichenfläche um das Spiel darzustellen
- nimmt die Eingaben des Spielers entgegen und steuert das Spiel
- nimmt die Rückmeldungen des Spiels entgegen und verarbeitet diese

5. Datenbank

Die Datenbank sichert Daten zu jedem abgeschlossenen Spiel und wenn das Spiel gespeichert wird. Es handelt sich um eine lokale SQLLight Datenbank, deren Datei sich im Ausführungsverzeichnis befinden muss.



Attribute (nicht alle):

- moves: Anzahl der Züge
- startplayer: nicht die oid des Spielers sondern 0 für player0 und 1 für player1
- movehistory: text, dessen Ziffern für die Spalten stehen, in die Spielsteine geworfen worden
- level: nur relevant, wenn iscpu gesetzt
- iscpu: Das attribut wird gegen NULL und nicht gegen seinen Wert abgefragt
- finished: 0 – nicht beendet, 1 – player 0 ist der Gewinner, 2 – Unentschieden.

Zur Sicherung der Dateien werden nur zwei Tabellen benötigt. Auf zwei Tabellen für abgeschlossene Spiele und Spiele zum weiterspielen für die Use Cases Highscore ansehen und Spiel laden wurde verzichtet. Stattdessen werden die relevanten Spiele an Hand des Attributs finished für die jeweiligen Fälle gefiltert.

Zwar ist 'nimmt teil' formal eine n zu m Beziehung, aber diese kann auch über über den doppelten Fremdschlüssel player0, player1 in games hergestellt werden und durch ein Kreuzprodukt der Tabelle player mit sich selbst:

Abfrage für die Highscore

```
select name, name1, moves from
  players,
  (select oid, name as name1 from players) as players1,
  games
where players.oid = games.player0 and players1.oid = games.player1
order by moves
```

Durch das temporäre Umbenennen der Tabelle players im Kreuzprodukt ist es möglich beide Spielernamen zu erhalten.

Das Datum wird sowohl über die date() funktion eingeschrieben als auch über date(...) und time(...) in zwei getrennte Werte ausgelesen:

Abfrage für das Laden der Spiele:

```
select date(datum), time(datum), name, name1 from
  players,
  (select oid, name as name1 from players) as players1,
  games
where players.oid=games.player0
  and players1.oid=games.player1
  and finished=0 and movehistory not null
order by datum desc
```

6. Spielbrett Designs

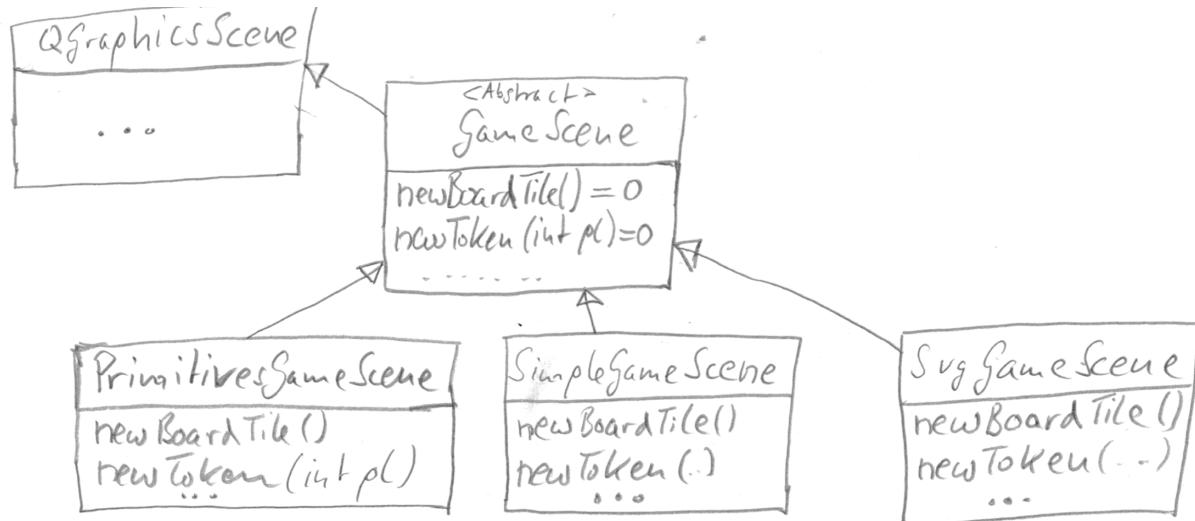
Das Spielbrett kann in 3 verschiedenen Designs angezeigt werden. Das Spielbrett wird mit Hilfe eines QGraphicsScene Objekts auf der Zeichenfläche einer QGraphicsView gezeichnet.

Um die verschiedenen Designs möglichst generisch zu implementieren gibt es eine abstrakte Elternklasse

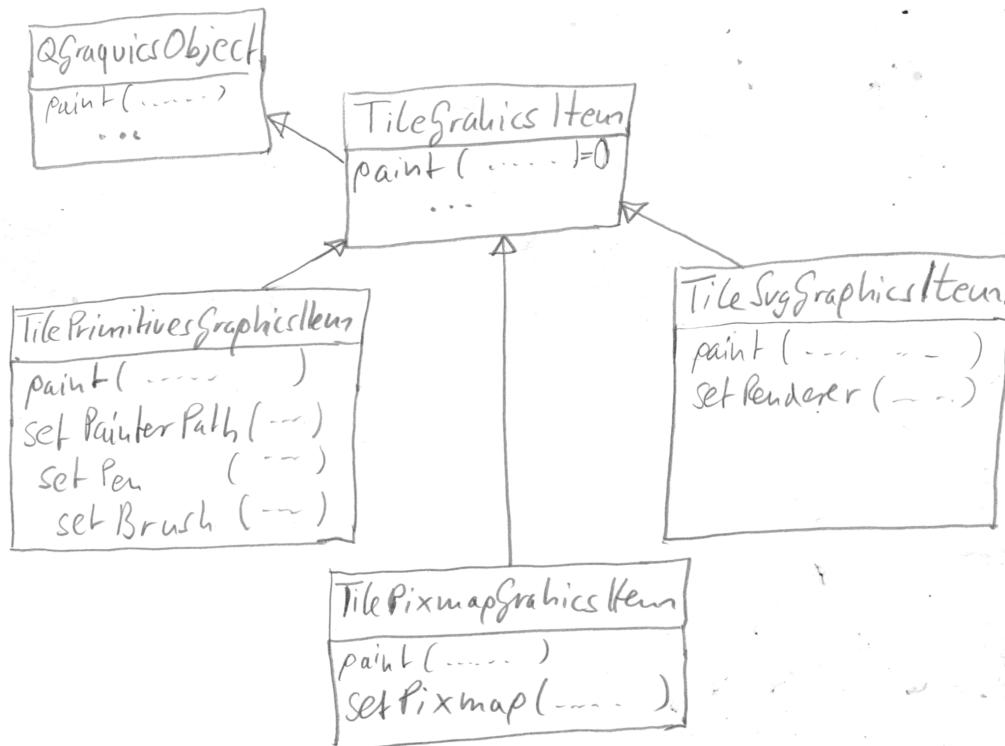
GameScene, die bereits die Positionierung der Spielsteine, sowie Animation ausführen kann. Nur die Objekte die gezeichnet werden sind noch nicht implementiert.

Das Spielfeld selbst wird aus quadratischen Kacheln zusammengesetzt, die genau einen Spielstein aufnehmen können und auch die Spielsteine selbst stellen Kacheln dar. Daher erben alle Objekte, die in einer GameScene gezeichnet werden von dem Eltern Objekt TileGraphicsItem.

Implementationen der GameScene muessen nun nur die Methoden implementieren, die die verschiedenen Kacheln erzeugen.



Für jedes Szenendesign gibt es auch eine Kachelklasse, die wiederum die Zeichenmethode paint(...) überschreiben muss.



Die Settermethoden in den verschiedenen TileGraphicsItems, setzen jeweils eine Referenz auf die Resource, die zum Zeichnen des Items notwendig ist, so dass nicht pro Tile eine Kopie erstellt wird, sondern Tiles gleichen Typs eine Resource teilen.