Process

I decided to develop the model and the UI in tandem rather than starting solely with the model. This allowed me to test the model visually.

My first goal was to have a pass-and-play version of the game.

The first day I worked on the project I developed the UI and model up to the point that was swapping was possible. I made the recyclerview for the player's tiles and a button for swapping tiles, as well as developing the onClickListeners for selecting tiles.

Next I made an expandable board and made tiles placeable on this board, followed by implementing a very basic check for valid tiles. As time went on, I added more and more checks in the model until placements followed the rules perfectly.

Once a pass-and-play game was possible, I made my server, using the chat client server from our lectures as my guideline and inspiration.

After many, many exceptions being thrown I finally got the multiplayer functionality working for games of only two players.

Now I made a results screen and implemented it for both the local and multiplayer versions of the game.

Now that the app was basically working, I added sounds to polish it off and gave it to my brother and girlfriend to test. Immediately, my girlfriend tried to drag the tiles that I only had onCLickListeners attached to. Thus my final addition to the app was allowing drag-and-drop functionality on my tiles.

Design decisions

I have a separate board activity for networked games. I could have had the same activity for both local and networked games but I decided for my own ease of use I would separate them. There is therefore a lot of repeated code shared by these two activities but reading the separate classes is a lot easier. Some of the shared methods were removed and put in a Controller class to minimise the repeated code.

After semester test 2 I decided to implement my own version of the Tile class that we made use of in the practical version of our test.

What worked – and what didn't

- I was very keen on the idea of an expandable grid, but wasn't sure how to go about it. I wanted to work with a 2D array as these are familiar to me, so I though I'd store the grid in a

215 x 215 array and only display the necessary tiles on screen. However, this took way too long to generate and so I scrapped that idea.

- ✓ After our second semester test, I decided to implement the Tile class we worked with during the practical section of the test. This worked a lot better and allowed my grid to expand as wanted.

- ▪ I wanted zoom functionality on my board. I added the appropriate buttons and achieved some functionality, but I believe the nature of the layouts and views I was using was not conducive to easy zooming. Pressing these buttons messed up the whole UI, making it unplayable.

- ✓ I put the grid in a horizontal scroll view in a vertical scroll view. This allows players to see the whole board upon scrolling.

- ▪ I tried to use Instants to get the time between someone starting an online game and the next player joining. This this did not work out for me like I wanted it to as I kept getting a difference of 6 or 7 seconds between players joining, even though I waited over 30 seconds to join.

- ✓ I instead made it so that it doesn’t matter how long you take to join, as long as no moves have been made you can still join the game.

<u>The connection process</u>

When ‘Connect’ is pressed, a connection is made to the server using the provided server address. If this connection is successful, the connection is cancelled, the waiting screen opens, and the connection is re-established.

The client send a WaitingForGame message to the server.

The server adds the client to the first Game that has less than 4 people and that has not had any moves made yet. If the model associated with that Game has at least 2 people, the model is sent to all players in that Game.

Upon receiving a model from the server, the board is populated using this model’s information.

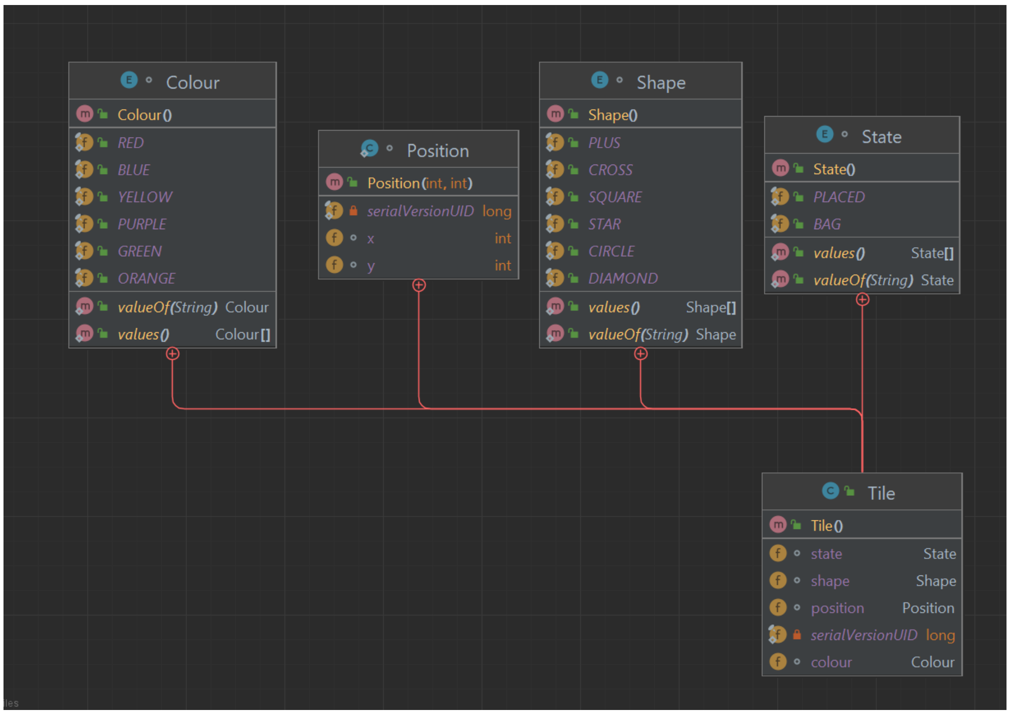
Every time a tile is placed or the turn is ended, a TurnFinishedMessage is send to the server, which then sends out this model to all players.

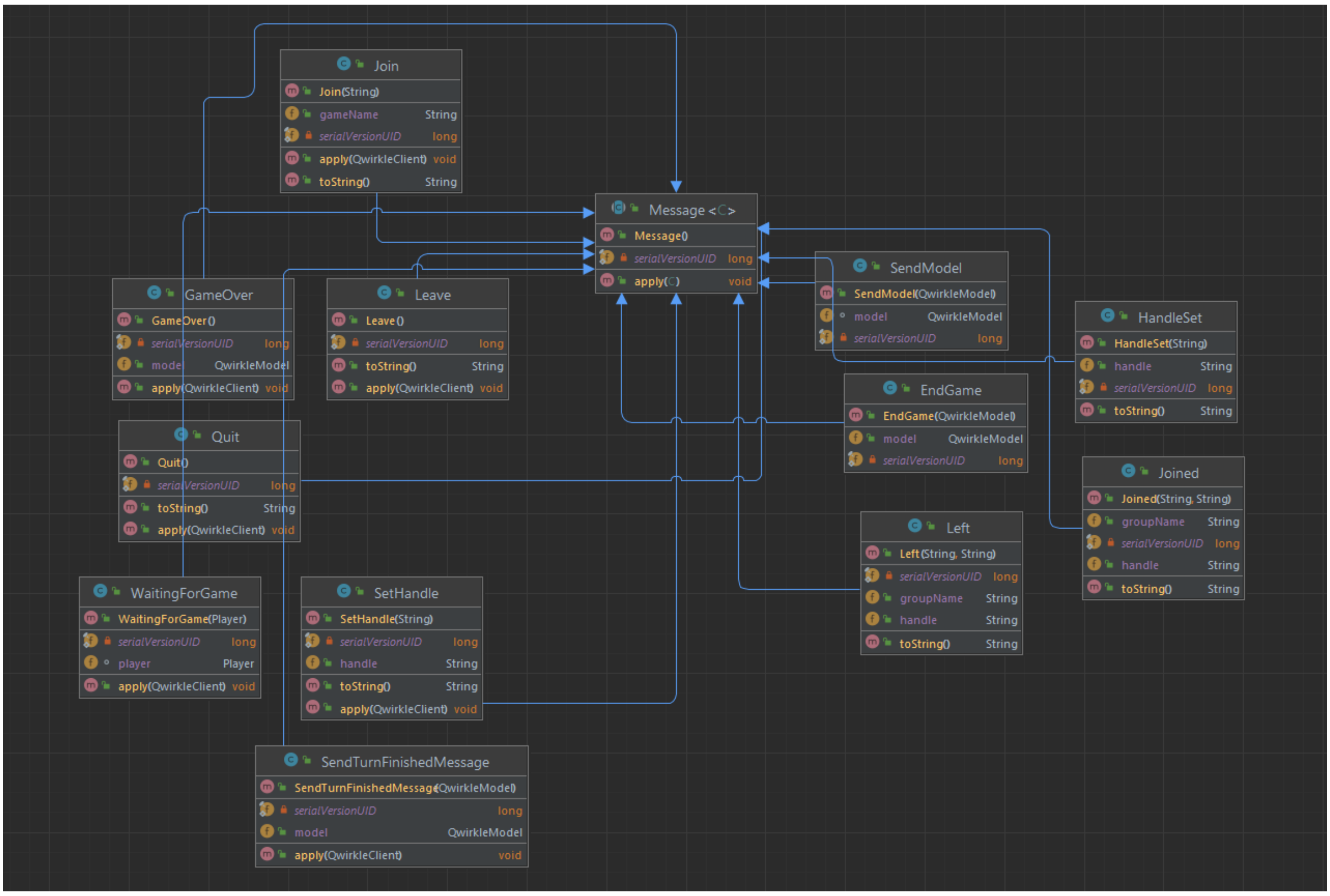
If the game is over at the end of a turn, the client send a GameOver message to the server.

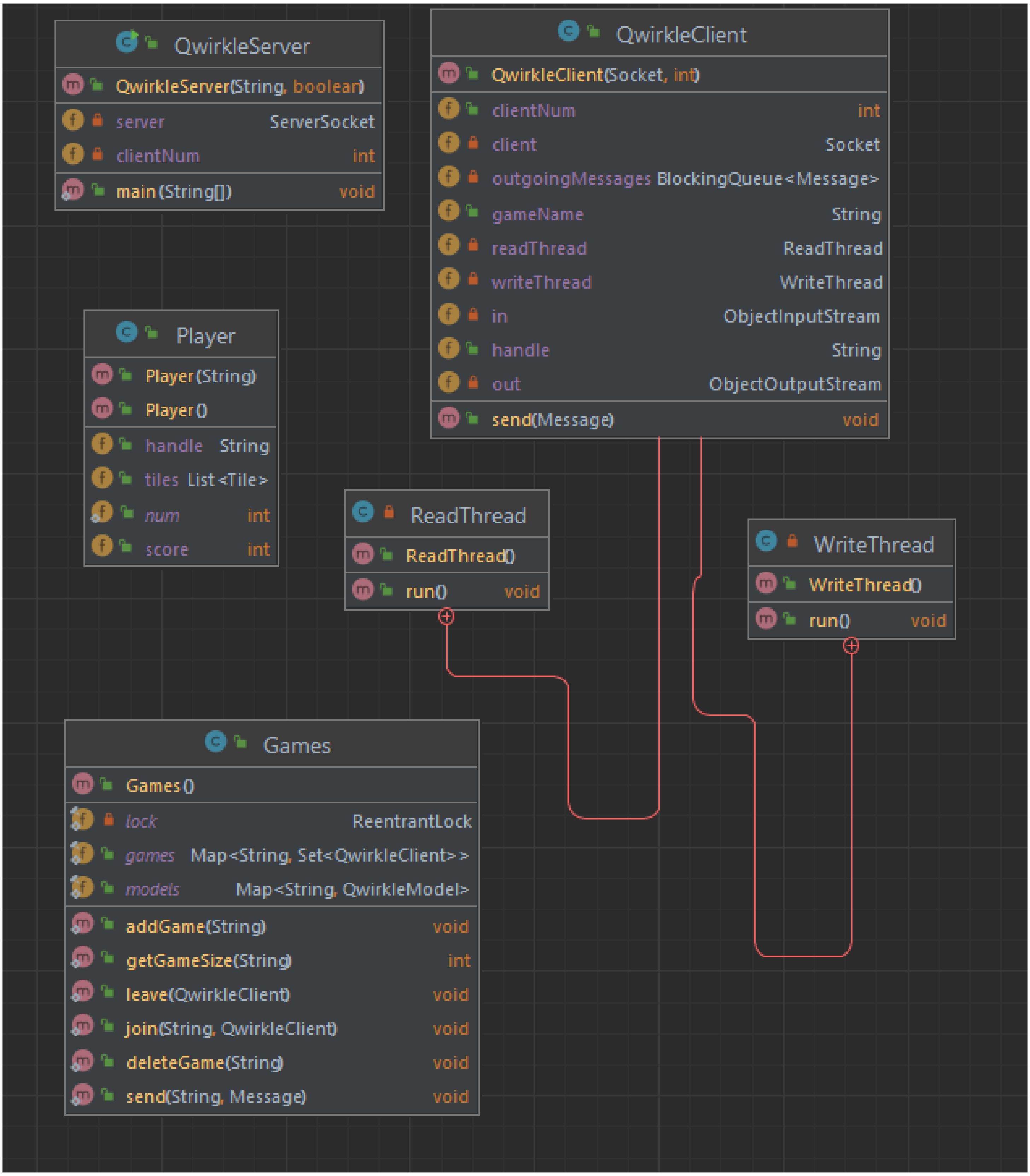
Upon receiving a GameOver message, the server notifies all the clients of that Game and those clients are sent to the results activity.

UML Diagrams

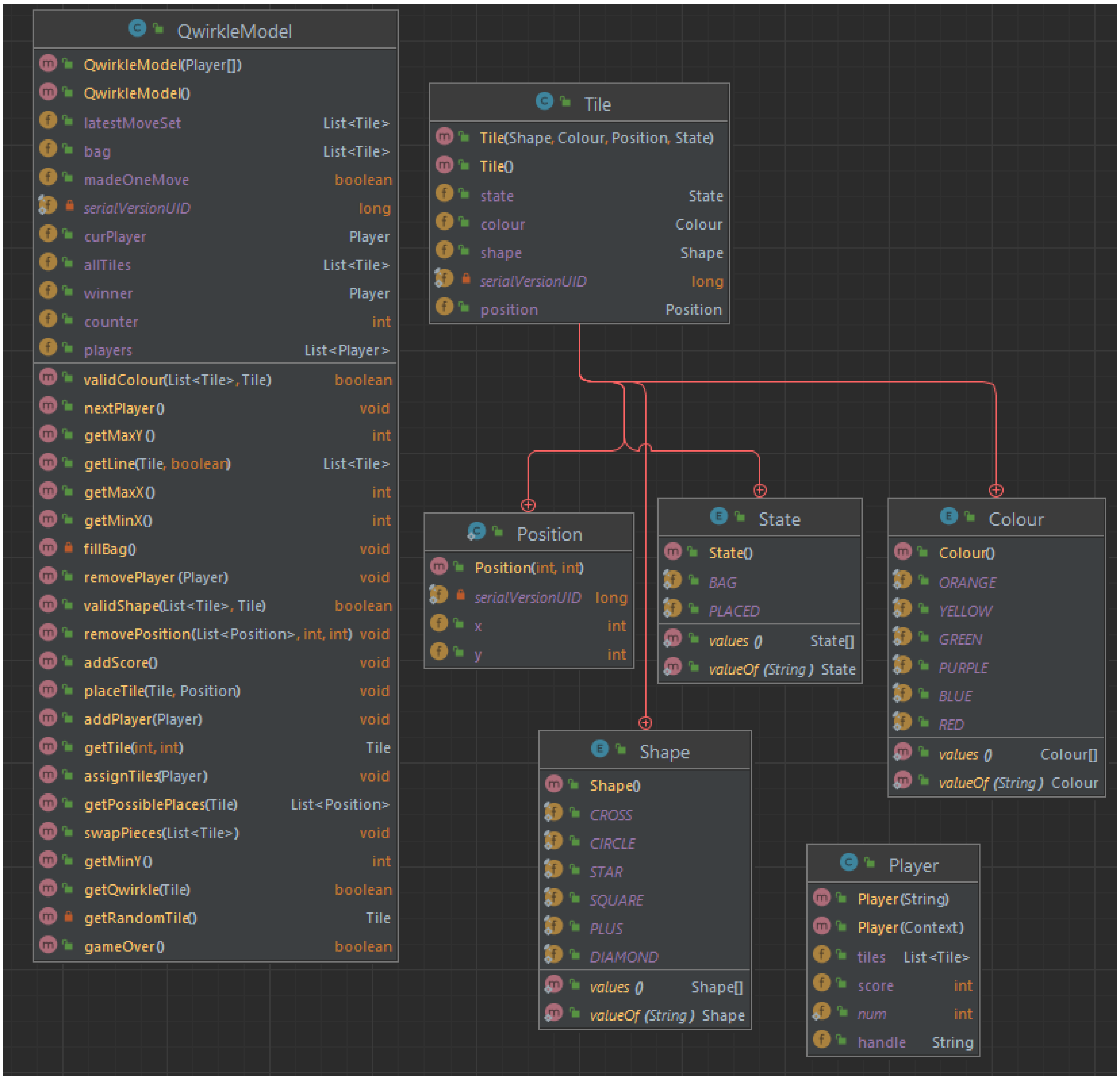
Server






QwirkleServer
QwirkleServer(String, boolean)
server ServerSocket
clientNum int
main(String[]) void
QwirkleClient
QwirkleClient(Socket, int)
clientNum int
client Socket
outgoingMessages BlockingQueue<Message>
gameName String
readThread ReadThread
writeThread WriteThread
in ObjectInputStream
handle String
out ObjectOutputStream
send(Message) void
Player
Player(String)
Player()
handle String
tiles List<Tile>
num int
score int
ReadThread
ReadThread()
run() void
WriteThread
WriteThread()
run() void
Games
Games()
lock ReentrantLock
games Map<String, Set<QwirkleClient>>
models Map<String, QwirkleModel>
addGame(String) void
getGameSize(String) int
leave(QwirkleClient) void
join(String, QwirkleClient) void
deleteGame(String) void
send(String, Message) void

Client



```
QwirkleModel
QwirkleModel(Player[])
QwirkleModel()
latestMoveSet                            List<Tile>
bag                                      List<Tile>
madeOneMove                              boolean
serialVersionUID                         long
curPlayer                                Player
allTiles                                 List<Tile>
winner                                   Player
counter                                  int
players                                  List<Player>
validColour(List<Tile>, Tile)            boolean
nextPlayer()                             void
getMaxY()                                int
getLine(Tile, boolean)                   List<Tile>
getMaxX()                                int
getMinX()                                int
fillBag()                                void
removePlayer(Player)                     void
validShape(List<Tile>, Tile)             boolean
removePosition(List<Position>, int, int) void
addScore()                               void
placeTile(Tile, Position)                void
addPlayer(Player)                        void
getTile(int, int)                        Tile
assignTiles(Player)                      void
getPossiblePlaces(Tile)                  List<Position>
swapPieces(List<Tile>)                   void
getMinY()                                int
getQwirkle(Tile)                         boolean
getRandomTile()                          Tile
gameOver()                               boolean
```

```
Tile
Tile(Shape, Colour, Position, State)
Tile()
state              State
colour             Colour
shape              Shape
serialVersionUID   long
position           Position
```

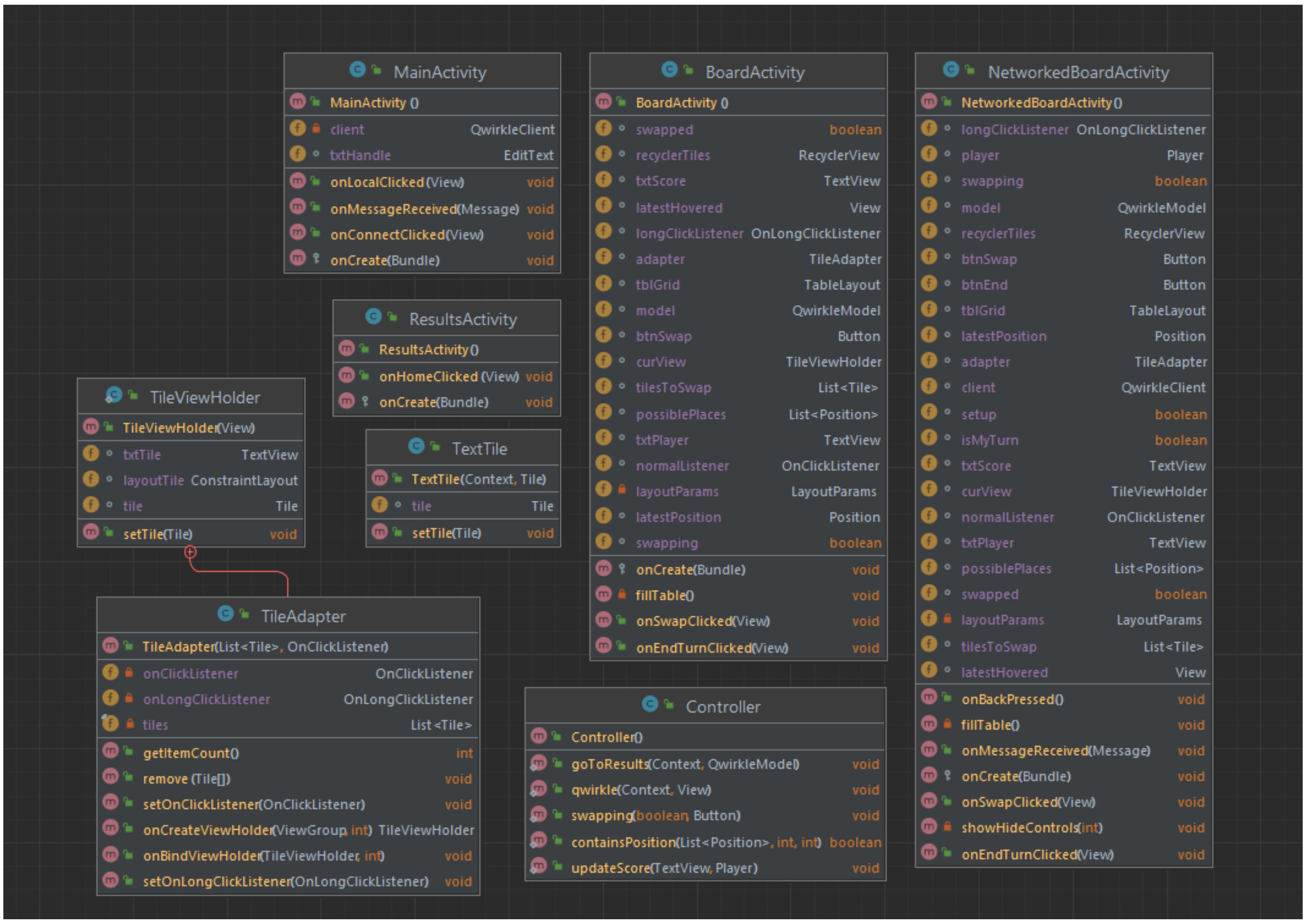
```
Position
Position(int, int)
serialVersionUID   long
x                  int
y                  int
```

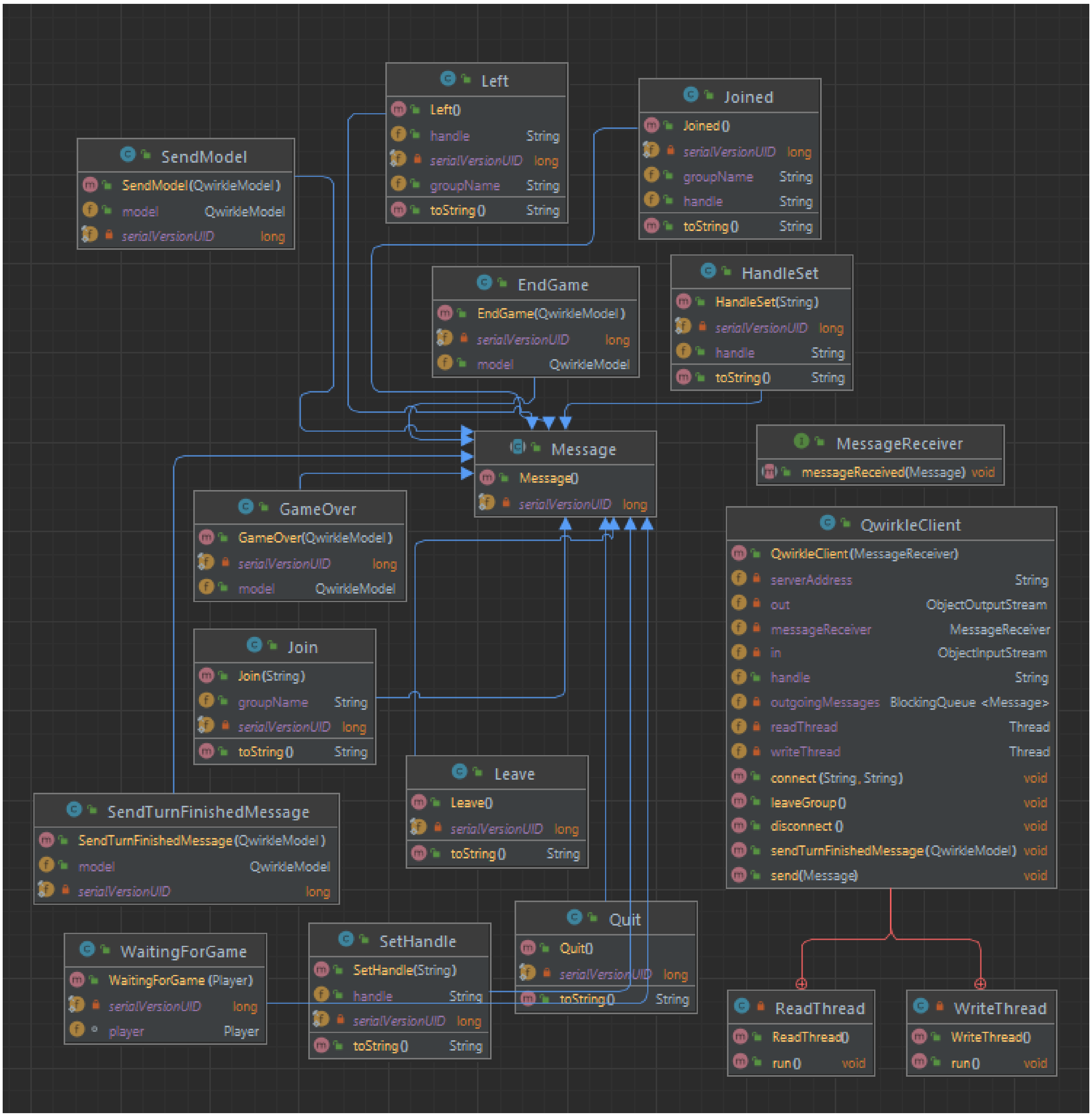
```
State
State()
BAG
PLACED
values ()          State[]
valueOf (String)   State
```

```
Colour
Colour()
ORANGE
YELLOW
GREEN
PURPLE
BLUE
RED
values ()          Colour[]
valueOf (String)   Colour
```

```
Shape
Shape()
CROSS
CIRCLE
STAR
SQUARE
PLUS
DIAMOND
values ()          Shape[]
valueOf (String)   Shape
```

```
Player
Player(String)
Player(Context)
tiles     List<Tile>
score     int
num       int
handle    String
```


MainActivity
MainActivity ()
client QwirkleClient
txtHandle EditText
onLocalClicked(View) void
onMessageReceived(Message) void
onConnectClicked(View) void
onCreate(Bundle) void
ResultsActivity
ResultsActivity()
onHomeClicked(View) void
onCreate(Bundle) void
TileViewHolder
TileViewHolder(View)
txtTile TextView
layoutTile ConstraintLayout
tile Tile
setTile(Tile) void
TextTile
TextTile(Context, Tile)
tile Tile
setTile(Tile) void
TileAdapter
TileAdapter(List<Tile>, OnClickListener)
onClickListener OnClickListener
onLongClickListener OnLongClickListener
tiles List<Tile>
getItemCount() int
remove (Tile[]) void
setOnClickListener(OnClickListener) void
onCreateViewHolder(ViewGroup, int) TileViewHolder
onBindViewHolder(TileViewHolder, int) void
setOnLongClickListener(OnLongClickListener) void
BoardActivity
BoardActivity ()
swapped boolean
recyclerTiles RecyclerView
txtScore TextView
latestHovered View
longClickListener OnLongClickListener
adapter TileAdapter
tblGrid TableLayout
model QwirkleModel
btnSwap Button
curView TileViewHolder
tilesToSwap List<Tile>
possiblePlaces List<Position>
txtPlayer TextView
normalListener OnClickListener
layoutParams LayoutParams
latestPosition Position
swapping boolean
onCreate(Bundle) void
fillTable() void
onSwapClicked(View) void
onEndTurnClicked(View) void
Controller
Controller()
goToResults(Context, QwirkleModel) void
qwirkle(Context, View) void
swapping(boolean, Button) void
containsPosition(List<Position>, int, int) boolean
updateScore(TextView, Player) void
NetworkedBoardActivity
NetworkedBoardActivity()
longClickListener OnLongClickListener
player Player
swapping boolean
model QwirkleModel
recyclerTiles RecyclerView
btnSwap Button
btnEnd Button
tblGrid TableLayout
latestPosition Position
adapter TileAdapter
client QwirkleClient
setup boolean
isMyTurn boolean
txtScore TextView
curView TileViewHolder
normalListener OnClickListener
txtPlayer TextView
possiblePlaces List<Position>
swapped boolean
layoutParams LayoutParams
tilesToSwap List<Tile>
latestHovered View
onBackPressed() void
fillTable() void
onMessageReceived(Message) void
onCreate(Bundle) void
onSwapClicked(View) void
showHideControls(int) void
onEndTurnClicked(View) void



Extra functionality

The game is fully translatable to Afrikaans.