

ARCHITEKTUR

Marten Buchmann, Katinka Feltes, Carl Gathmann, Helen Kuswik

Erläuterungen zur Architektur

Gruppe 5

Grundlegendes

Model-View-Controller gewählt und abgewandelt

- CliController regelt den Spielablauf für CLI-View
- GuiController regelt den Spielablauf für GUI-View
- Model entspricht der Klasse Game (und weiteren Klassen, auf die Game zugreift), in der die Funktionalität des Spiels implementiert ist

→ Gut erweiterbar (z.B. Zusatzfeature „Spielmodi“ war kein Problem einzubinden)

Interface PlayerLogic bei Erstellung der KI kreiert, dass bei Spielentscheidungen gefragt wird

- Entscheidungen funktionieren unterschiedlich, je nachdem ob Spieler AI oder Human ist
- KI: Algorithmus entscheidet (s. Abschnitt „KI“ weiter unten)
- Human: Input des Spielers

Vollständiges JavaDoc, Architekturdokumentation und hohe Test-Abdeckung → Gute Wartbarkeit

Wahlfeatures

Tokens (Wegeplättchen): als Interface implementiert

- so können dem Spielfeld generell Tokens zugeordnet werden
- haben alle die gleichen Grundfunktionen (Action, Name, collectable)

Speichern/Laden: Interface Serializable implementiert, um Speichern zu vereinfachen

Orakelpriesterin: Feld in der Klasse Game

Highscores: String-Liste mit den Highscores, die aus einer Datei eingelesen wird. Die Liste wird ggf. angepasst und die Datei wird überschrieben, wenn das Spiel vorbei ist.

Spielernamen und Farben: Spielernamen kann man bei uns nur für Menschen auswählen, damit das Spiel schneller gestartet werden kann. Statt Farben kann man sich ein Symbol aussuchen.

Ziehen vom Abwurfstapel und Koboldplättchen

Andere wichtige Erläuterungen

Exceptions werden bei uns spezifischer ausgegeben:

- „!Invalid move“:
 - falsche Zahl: „Please enter a number between x and y“
 - keine Zahl: „Input must be an Integer“
 - falsche Karte: „There is no card greater than 10“ / „Please enter one of the options above“
 - Zustimmung: „Please enter either y or n“
 - Eingabe von Stapel/Hand: „Please enter either r,g,b,o or p“ / „Please enter either r,g,b,o,p or h (hand)“
- „!Move not allowed“:
 - Karte ziehen: „You fool: this pile is empty“ / „You can't draw a card you just trashed“
 - Karte spielen: „Card is not in hand“ / „Card does not fit into pile“ / „This figure can't move this far“
 - Karte abwerfen: „Card is not in hand“ / „Card is null or different color than the pile“
 - Spiral-Token: „You can not go to the field you came from“
 - Goblin-Token: „Card is not in hand“
 - Orakelpriesterin: „The zombie cannot move this far“

Die Wiederholung von Eingaben haben wir weggelassen, damit die Ausgabe nicht überfüllt wird. Sonst müsste man sehr weit scrollen, um sich das Spielfeld anzugucken.

KI

Gruppe 5

Vor- überlegungen

KI

weit nach vorne	Punkte + Schritte (am Ende keine neue Fig) weil negativer Gewinn ⇒ Farbe + Fig	# Schritte + 2 Punktegewinn 0,5
Differenz zu gelegten Zahlen	⇒ methode: beste Differenz beim legen einer Farbe?	- Differenz 2,5
Tokens: Goblin Mirror Skullpoint Spiderweb Spiral Wishing	Nur gut, wenn schon andere Goblins oder schlechte Karten/Stapel So viel wert wie Stone Plot Punkte wert 1-4 So 4 Schritte weiter im Durchschnitt Ok → schwer zu machen tho ⇒ egal von Wertung her Super, wenn nicht schon 5	+ 10 wenn #Goblin 2 und not played + 3 wenn #G18 not Played oder schlechte Karten sonst + 0 +/- currentToken verlue + Plot + 4 (weil auch noch neues Token dazu) → Gut + 0 + 2,8 · mirror weil 8

nicht aus
dem Feld raus

mehr wert
weil Token sonst
zu mächtig

KI – Generelle Entscheidungen

Spielen oder Abwerfen? Spielen, wenn mehr als 2 Karten spielbar sind.

Goblin-Special spielen? Ja immer, denn es gibt nicht oft die Möglichkeit.

Nachziehstapel wählen: Vom Nachziehstapel, außer es gibt einen Abwurfstapel mit einer Karte, wo Differenz zu den passenden Played-Cards + #Handkarten der Farbe auf der Hand < 5 ist

Spiralposition wählen: Die letzten 5 Felder anschauen und das Feld wählen mit dem besten Token (nur nicht das, woher man kommt). Gibt es kein gutes Token, bleibt die Figur einfach stehen.

Goblin-Aktion wählen: Immer nutzen, denn Abwerfen von den Played-Cards schadet es ja auch nicht. Handkarte abwerfen, wenn nur noch 2 spielbar sind. Sonst den Stapel mit der schlechtesten Differenz (zu 0/10 je nach Richtung)

KI – Spielen einer Karte

KI geht jede Figur-Karten-Kombination durch und bewertet sie nach verschiedenen Faktoren:

- + Anzahl der Schritte, die die Figur nach vorne machen würde
- + Punktezuwachs durch die Bewegung
- + Wert des Tokens (siehe nächste Folie)
- Differenz zu der obersten Karte des Played-Cards-Pile (100 falls die Karte nicht passt)

Die beste Option, eine Figur zu bewegen, wird dann verglichen mit der besten Option den Zombie zu bewegen. Der Wert dafür wird berechnet, indem jede Zombie-Karten-Kombi durchgegangen wird:

- Falls keine eigene Figur erreichbar wäre = -100
- + sonst 5
- Differenz zu der obersten Karte des Played-Cards-Pile

→ Die gewählte Karte UND eine Figur oder Anzahl der Schritte für den Zombie

KI – Wert der Token

Wishingstone: 2,8 (Ø Punktegewinn durch einen Wishingstone) × #Mirrors

Mirror: Momentaner Wert aller gesammelten Tokens (da dieser verdoppelt werden würde)

Skullpoints: 1-4 je nach Wert

Spiderweb: Wert des Zuges den der Sprung bringt (Schritte, Feldpunkte, Wert des neuen Tokens)

Goblin: 10, wenn nur noch 4 Handkarten spielbar sind oder 2 Figuren auf einem Goblin stehen
3, wenn nur noch 3 Handkarten spielbar sind oder 1 Figur schon auf einem Goblin ist
0, sonst

Spiral: 1, da meistens schlecht





GUI

Gruppe 5

Vor-
überlegungen

DIRE HORROR LAND

Points: 2 Kl1: 3 Fred: 6

Tokens: 1x  2x  4x  1x 

Active Player: **Such A Great Player**

Played Cards:



+



-



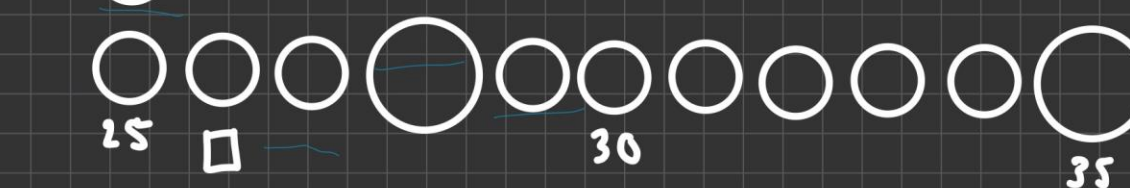
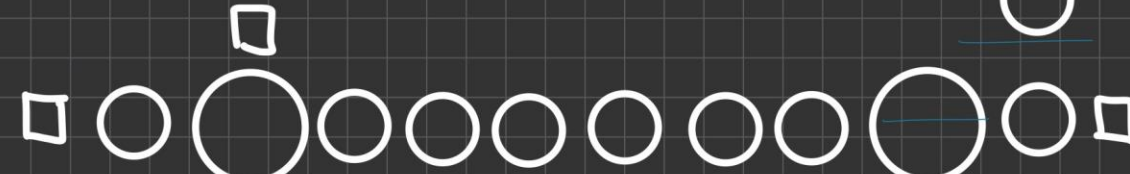
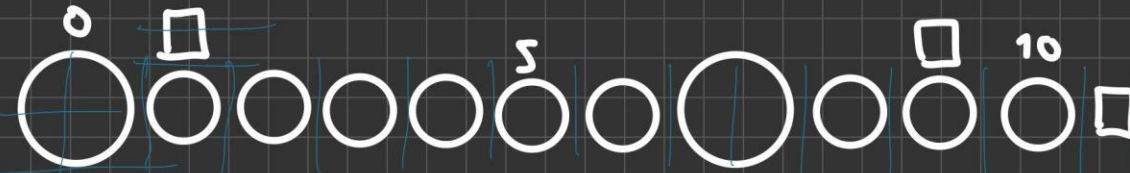
+ -



+



-



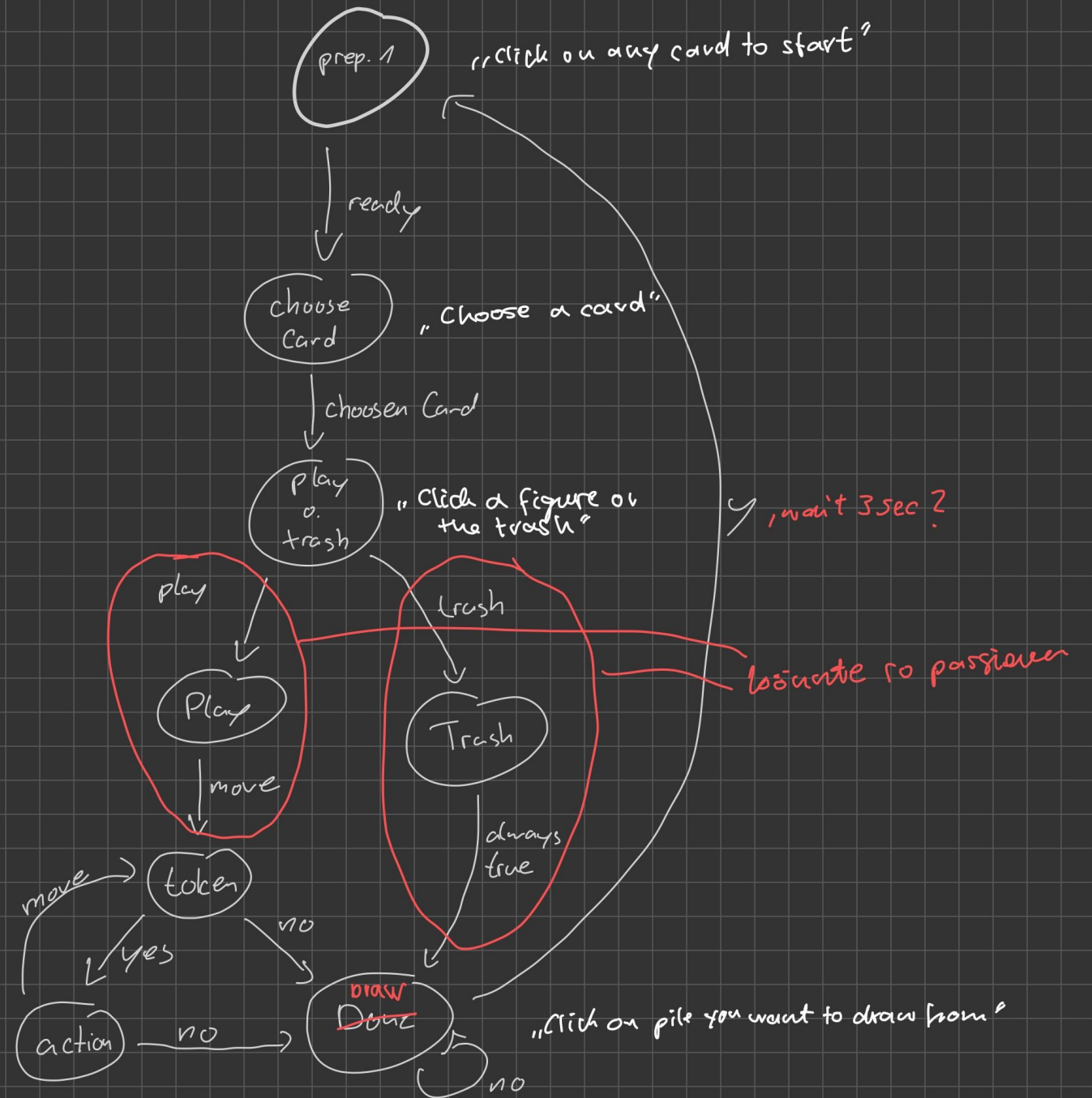
Hand
Cards:



Game Discarding
Piles:



Vorüberlegungen zu States



GUI

State Machine implementiert

- je nachdem in welchem State man sich befindet, sind andere Aktionen möglich
- Enum erstellt, um die States elegant zu speichern

Spielfeld:

- BorderPane für generelle Struktur des Spielfelds
- Felder im Grid Layout angeordnet → TilePanels, da sich hier die Spielfiguren automatisch auf den verfügbaren Platz verteilen
- Punkteleiste wurde aus Design-Gründen weggelassen (Übersichtlichkeit), stattdessen gibt es eine Score-Anzeige

States im Detail

PREPARATION

- Handkarten werden nicht angezeigt
- Variablen werden zurückgesetzt

CHOOSEHANDCARD

- Handkarten sind die einzigen gültigen Eingaben
- Chosen card wird gesetzt

TRASHORPLAY

- Chosen card kann überschrieben werden
- Spieler entscheidet: Zombie bewegen, Figur bewegen, Handkarte abwerfen

ORACLE

- Spieler entscheidet: Schrittzahl Zombie

SPIDERWEB, SPIRAL, GOBLIN

- Token Aktion wird ausgeführt

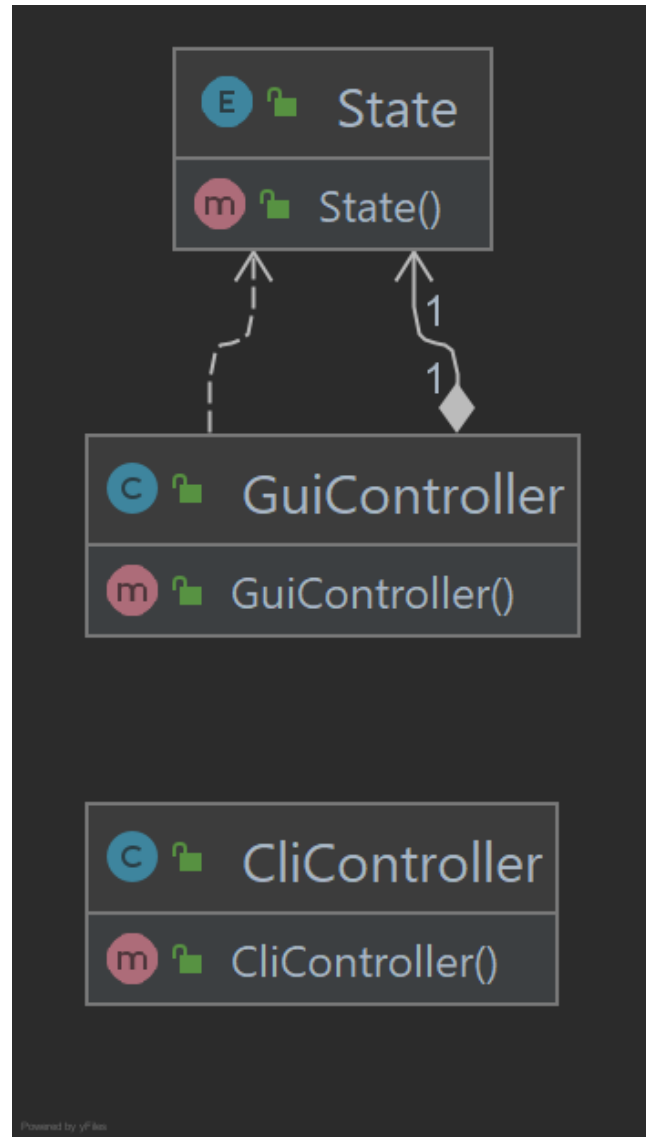
DRAW

- Spieler wählt: Wo nachgezogen wird (Nachziehstapel, Abgeworfene Karten)

Klassendiagramm

Gruppe 5

Klassendiagramme des Controllers



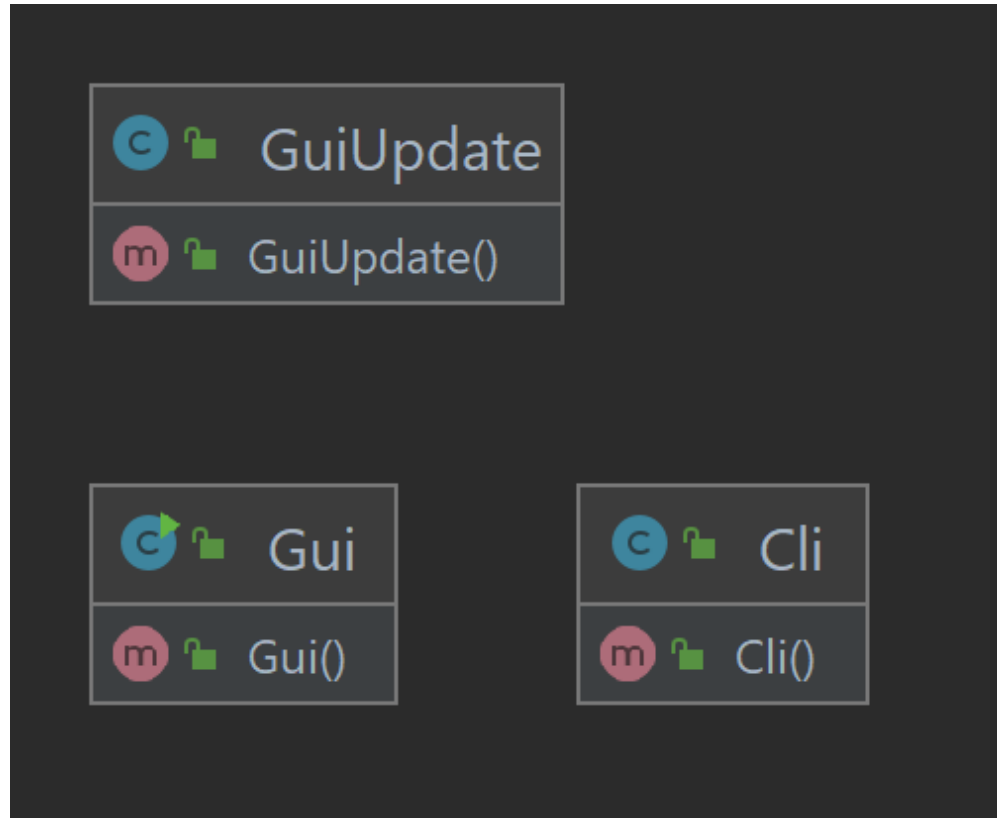
Dependencies der Controllerklassen

E State	
State()	
<i>valueOf(String)</i> State	
<i>values()</i> State[]	

C CliController	
CliController()	
doGoblinSpecialAction(Player)	void
doTokenGoblin(Player, Token)	void
doTokenSpiral(Player, Token)	void
drawOne(Player)	void
goblinChosenPile(Player, Token)	void
moveFigure(Player, Card)	void
moveOracle(Player, Card)	void
playCard(Player)	void
startGame()	void
takeTurn(Player)	void
trashCard(Player)	void
usingToken(Player)	void
waitForConfirmation(String, Player)	void

C GuiController	
GuiController()	
getIndex(String, String)	int
goblinAi(AI)	void
limitTo10Characters(KeyEvent)	void
loadFromSaved(ActionEvent)	void
loadGamemodes(ActionEvent)	void
loadHighscores(ActionEvent)	void
loadMenu(Event)	void
loadNewScene(Event, String, boolean, boolean)	void
onClick(MouseEvent)	void
removeGlow(MouseEvent)	void
save(ActionEvent)	void
saveGamemodes(ActionEvent)	void
setGlow(MouseEvent)	void
startGame(ActionEvent)	void
startTurn(MouseEvent)	void
takeTurnAI(AI)	void
updateAll()	void
updatePlayedCardsAndNames()	void

Klassendiagramme der View



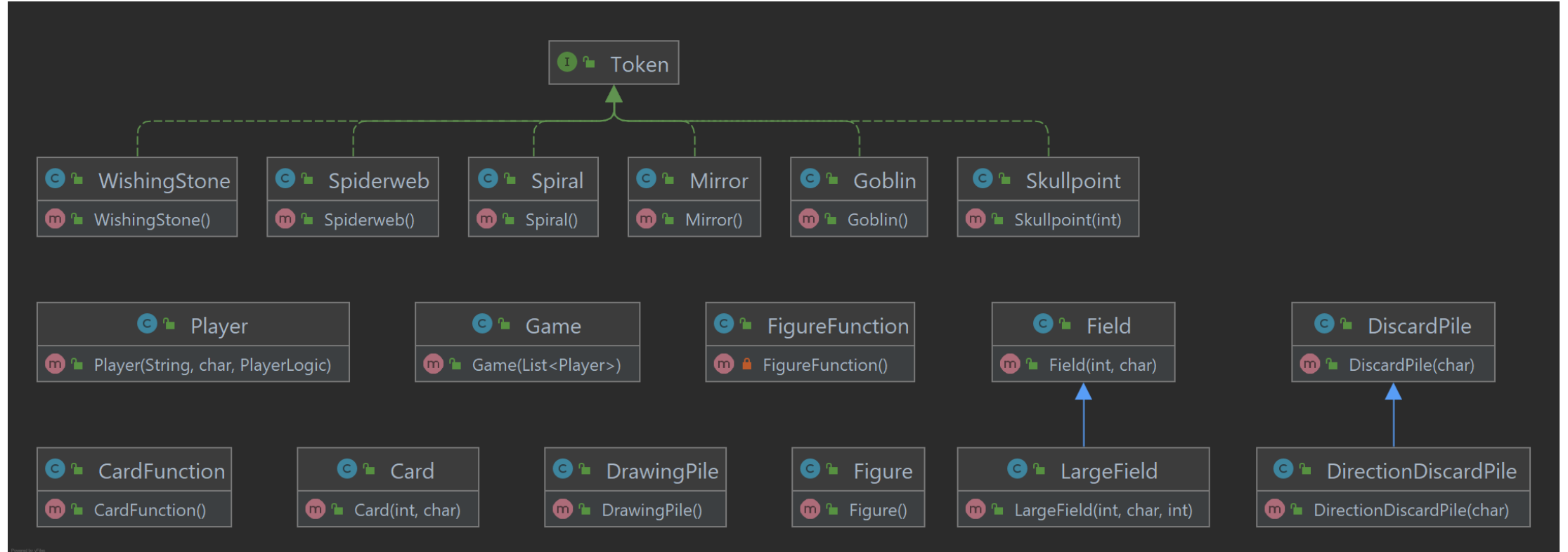
Klassen der View

Cli		
m	Cli()	
m	error(String)	void
m	inputPlayersNames(int)	String[]
m	out(String)	void
m	printBoardPart(int, int, Game)	void
m	printCurrentBoard(Game)	void
m	printDiscardingPiles(Game)	void
m	printHand(Player)	void
m	printResults(Game)	void
m	printTop(DirectionDiscardPile)	void
m	printTop(DiscardPile)	void
m	printTopCards(Player)	void
m	promptCardString(String)	String
m	promptColor(String)	char
m	promptInt(int, int, String)	int
m	promptPlayersChoice(String)	boolean

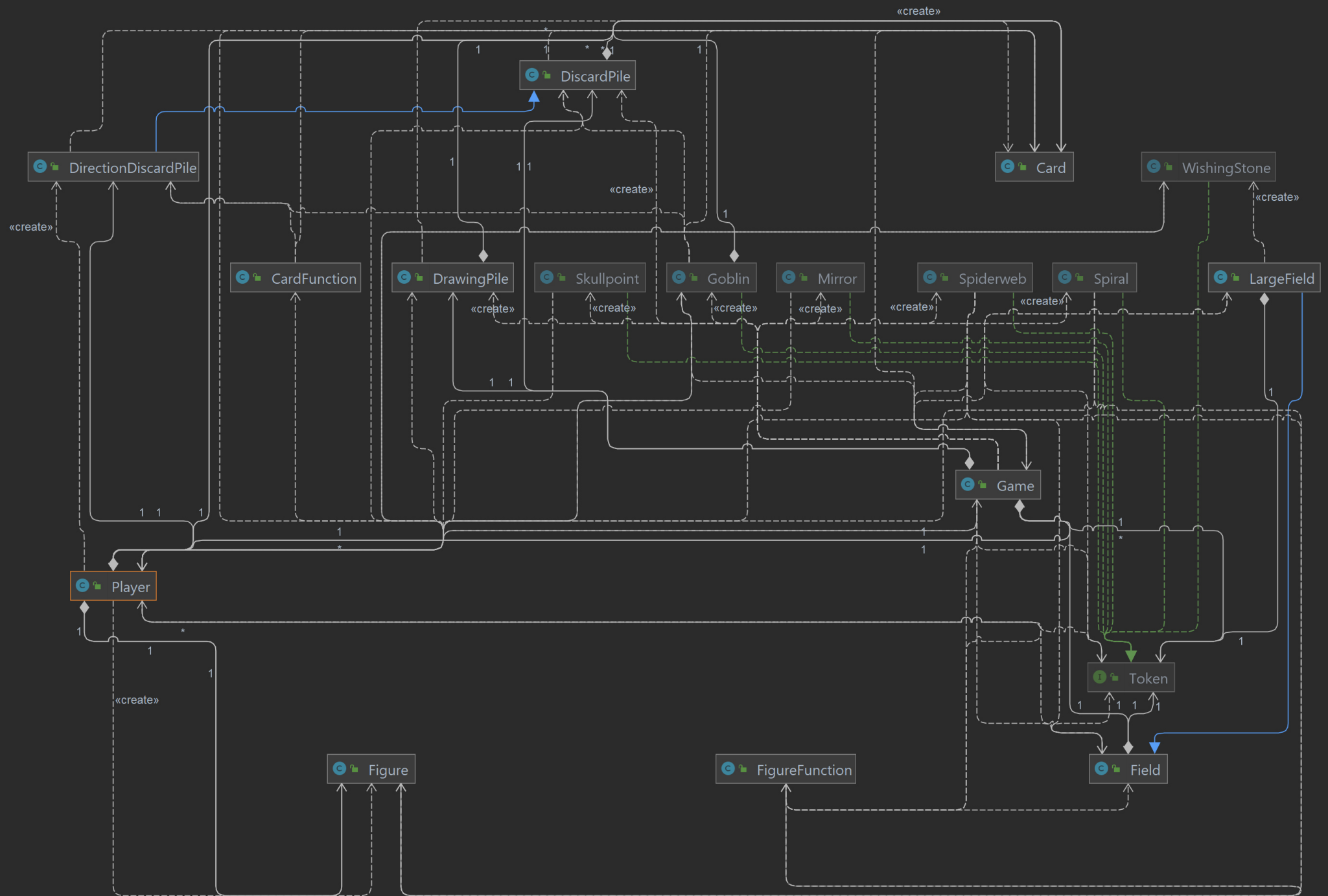
GuiUpdate		
m	GuiUpdate()	
m	classifyChildren(String, BorderPane)	List<Node>
m	createEndScores(List<Player>, BorderPane)	void
m	getAllChildren(Parent)	List<Node>
m	resetEffects(BorderPane)	void
m	updateCards(BorderPane, State, Player)	void
m	updateCollectedTokens(BorderPane, Player)	void
m	updateDiscardPiles(BorderPane, Game)	void
m	updateFigures(BorderPane, List<Player>, int)	void
m	updateScores(BorderPane, List<Player>)	void
m	updateTokens(BorderPane, Field[])	void

Gui		
m	Gui()	
m	main(String[])	void
m	start(Stage)	void

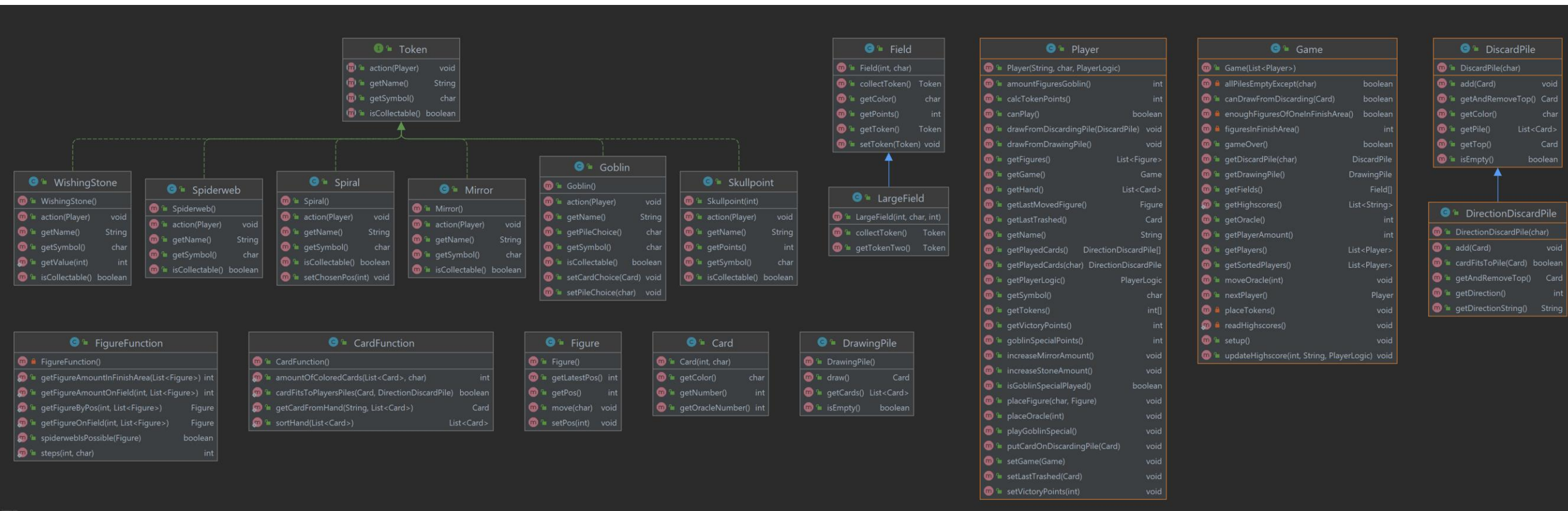
Klassendiagramm des Models



Klassen des Models

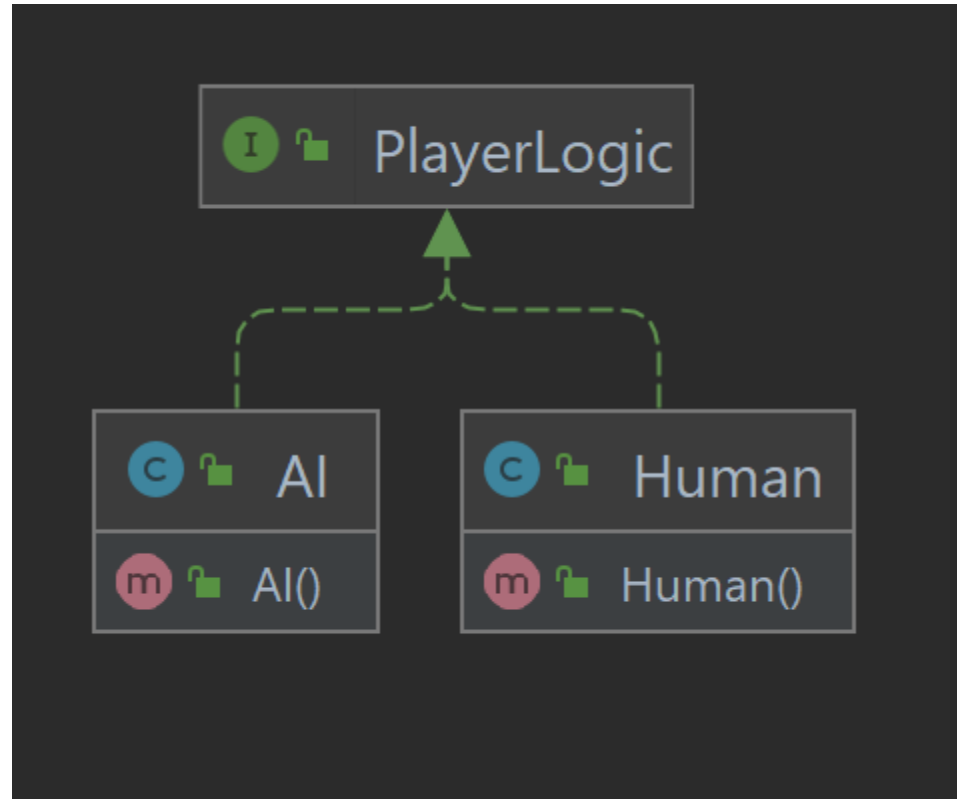


Depen-
dencies im
Model

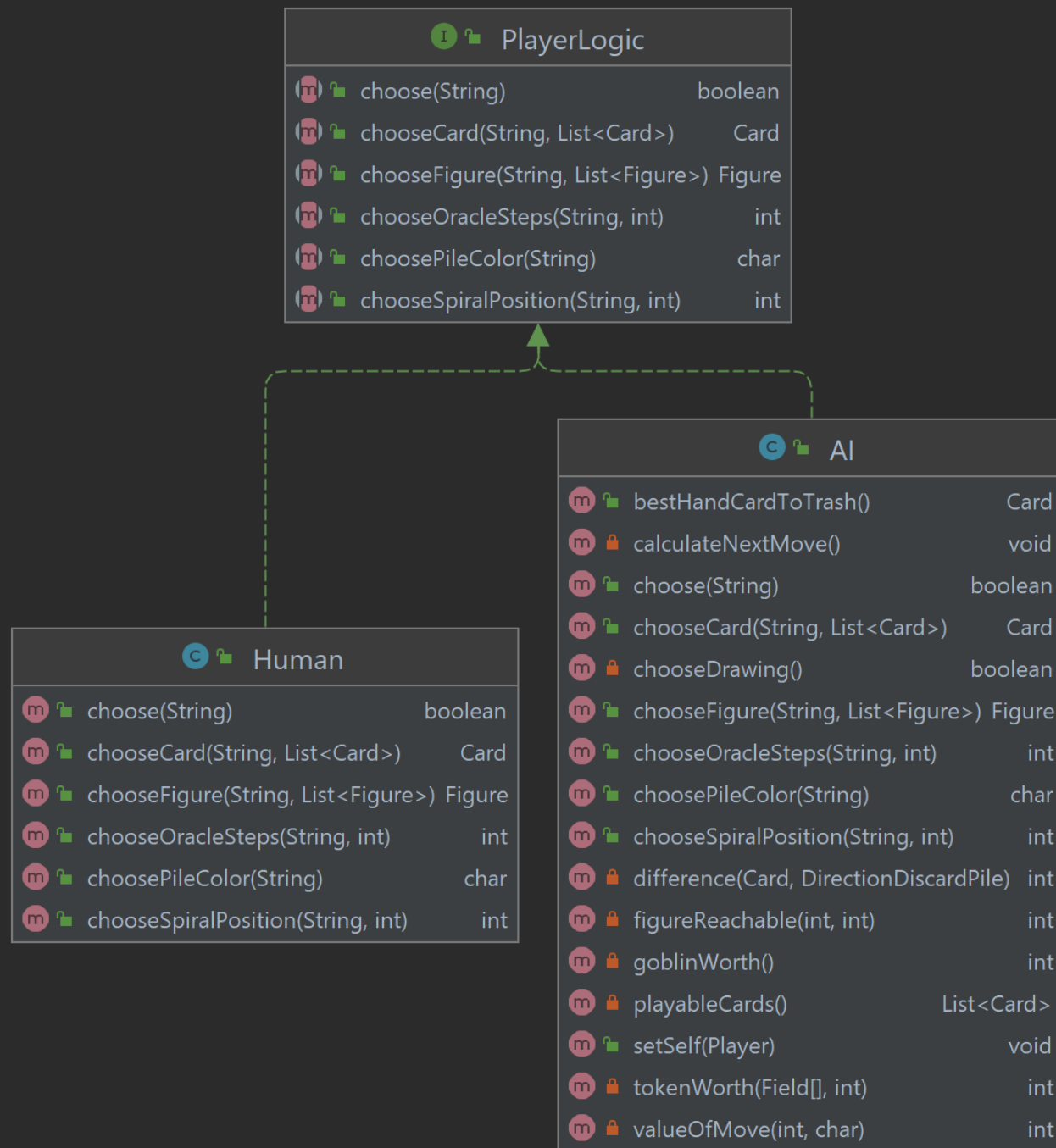


Methoden des Modells

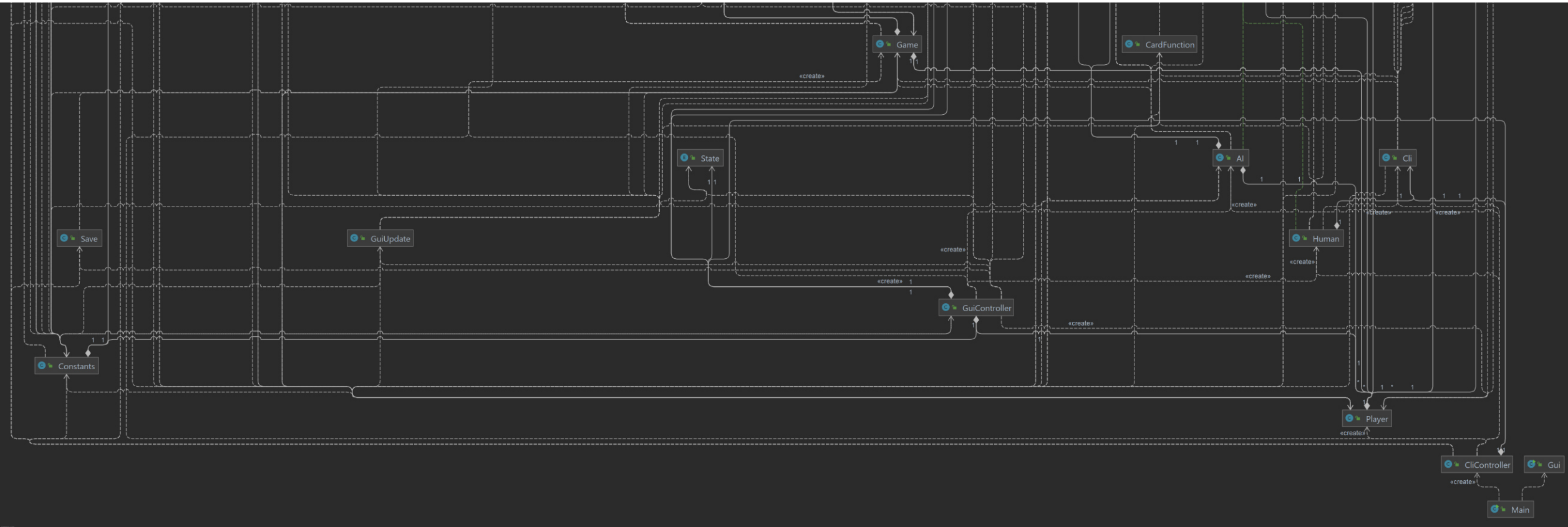
Klassen des PlayerLogic Packages



Klassen und Dependencies des Packages



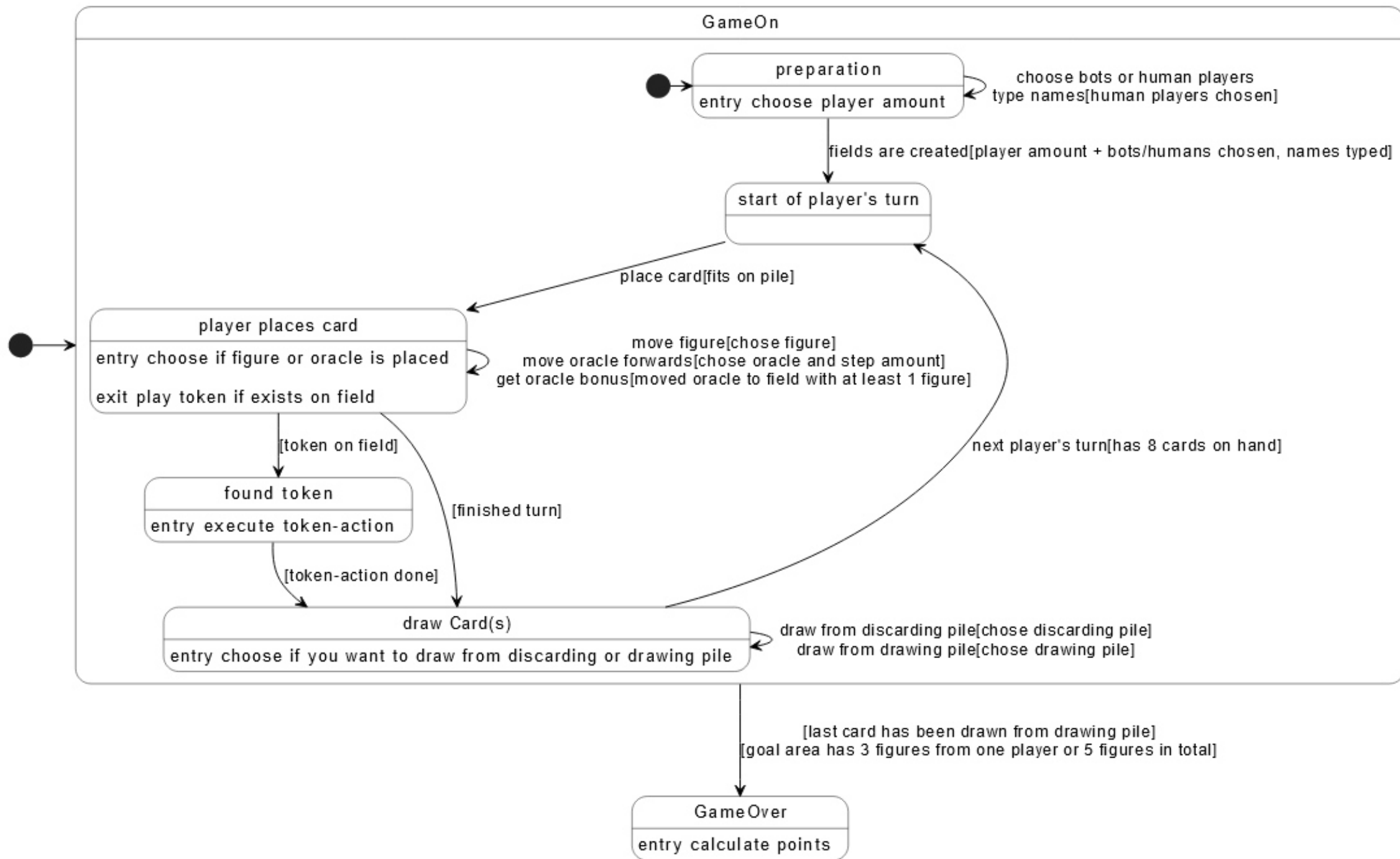
Restliche Klassen



Ausschnitt aus den gesamten Dependencies des Projekts

Zustandsdiagramm

Gruppe 5



Sequenzdiagramm

Gruppe 5

