PROG2 SoSe 2020
Chaos Reborn Reborn
Entwicklungsarbeit

Henning Kröger - 5093186
Joshua Christ - 5096129
Christian Vicky Temfac Djoken - 5103696
Jakob Spohler - 5094026

29. Oktober 2020

Inhaltsverzeichnis

1 Aufgabenstellung **11**

2 Anforderungsdefinition **11**

3 Entwurf **12**

4 Benutzungshinweise **14**
4.1 Das Hauptmenü . . . 14
4.2 Die Optionen . . . 15
4.2.1 Die Karteneinstellungen . . . 15
4.3 Das Pausemenü . . . 17
4.4 Ein lokales Spiel starten . . . 18
4.5 Ein Netzwerkspiel starten . . . 20
4.6 Das Spiel . . . 20
4.6.1 Eine Bewegung ausführen . . . 21
4.6.2 Einen Manakristall einsammeln . . . 21
4.6.3 Einen Angriff ausführen . . . 21
4.6.4 Eine Spielkarte verwenden . . . 22
4.7 Die Spielmechanik . . . 22
4.7.1 Spielziel . . . 22
4.7.2 Zauberstäbe . . . 22
4.7.3 Die Hand & Das Deck . . . 22
4.7.4 Der eigene Zauberer . . . 22
4.7.5 Angirffe & Bewegungen . . . 22
4.7.6 Zauberei . . . 25
4.7.7 Zaupersprüche . . . 26
4.7.8 Kreaturen . . . 26
4.7.9 Attribute . . . 27
4.7.10 Mana & Flux . . . 29

5 Anwendungsbeispiel **30**

6 Quelldateien **34**
6.1 Control . . . 34
6.1.1 Main.java . . . 34
6.1.2 Game.java . . . 34
6.2 Control.Actions . . . 36
6.2.1 Action . . . 36
6.2.2 AttackAction . . . 37
6.2.3 AttackMoveAction . . . 42
6.2.4 ManaPickupAction . . . 43
6.2.5 ManaPickupMoveAction . . . 44
6.2.6 MeleeAttackAction . . . 44

6.2.7 MoveAction . . . 46
6.2.8 RangedAttackAction . . . 48
6.3 Control.Controllers . . . 49
6.3.1 AnimationController . . . 49
6.3.2 GameController . . . 50
6.3.3 InputController . . . 52
6.3.4 MapController . . . 59
6.3.5 PlayerController . . . 62
6.3.6 SceneController . . . 63
6.3.7 SoundController . . . 66
6.3.8 TurnController . . . 67
6.4 Control.Network . . . 69
6.4.1 GameClient . . . 69
6.4.2 GameServer . . . 71
6.4.3 RecieveThread . . . 74
6.4.4 SendThread . . . 76
6.4.5 UserThread . . . 77
6.5 Control.Networks.Packets . . . 80
6.5.1 Packet . . . 80
6.5.2 PacketDecoder . . . 82
6.5.3 PacketType . . . 86
6.6 Control.SemiControllers . . . 87
6.6.1 Highlighter . . . 87
6.6.2 Selector . . . 90
6.7 Exceptions . . . 95
6.7.1 MapInvalidException . . . 95
6.7.2 MapTypeNotSupportedException . . . 96
6.7.3 UnexpectedCardException . . . 96
6.7.4 UnexpectedGameObjectException . . . 96
6.8 Model . . . 96
6.8.1 Player . . . 96
6.9 Model.Card . . . 100
6.9.1 CardAdapter . . . 100
6.9.2 CardEffectAdapter . . . 109
6.9.3 CardStack . . . 111
6.9.4 CardType . . . 113
6.9.5 Deck . . . 113
6.9.6 EffectTag . . . 115
6.9.7 GenericCardEffect . . . 115
6.9.8 Hand . . . 116
6.9.9 ICard . . . 121
6.9.10 ICardEffect . . . 122
6.9.11 Wand . . . 123
6.9.12 WandFactory . . . 126

6.10 Model.Card.AllCards . . . 127
6.10.1 CardFactory . . . 127
6.11 Model.Card.AllCards.Spells . . . 128
6.11.1 AttackSpell . . . 128
6.11.2 Spell . . . 131
6.12 Model.Card.AllCards.Spells.Chaos . . . 132
6.12.1 Vengeance . . . 132
6.13 Model.Card.AllCards.Spells.Law . . . 133
6.13.1 Decree . . . 133
6.13.2 DivineBow . . . 134
6.13.3 DivineShield . . . 135
6.13.4 DivineSword . . . 136
6.14 Model.Card.AllCards.Spells.Neutral . . . 138
6.14.1 MagicBolt . . . 138
6.14.2 WindWalker . . . 138
6.15 Model.Card.AllCards.Summons . . . 139
6.15.1 Summon . . . 139
6.15.2 SummonEffect . . . 141
6.16 Model.Card.AllCards.Summons.Chaos . . . 142
6.16.1 Hellhound . . . 142
6.16.2 Hydra . . . 143
6.16.3 Manticore . . . 143
6.16.4 Skeleton . . . 144
6.16.5 Vampire . . . 145
6.16.6 Zombie . . . 145
6.17 Model.Card.AllCards.Summons.Law . . . 146
6.17.1 IcarusTower . . . 146
6.17.2 Paladin . . . 146
6.17.3 Pegasus . . . 147
6.17.4 SapphireDragon . . . 147
6.18 Model.Card.AllCards.Summons.Neutral . . . 148
6.18.1 AirElemental . . . 148
6.18.2 Eagle . . . 149
6.18.3 Giant . . . 149
6.18.4 Lion . . . 150
6.19 Model.Card.Wands . . . 150
6.19.1 AirWand . . . 150
6.19.2 CreatureWand . . . 151
6.19.3 RandomWand . . . 153
6.19.4 UndeadWand . . . 153
6.20 Model.Game . . . 154
6.20.1 FluxMeter . . . 154
6.21 Model.GameObject . . . 156
6.21.1 Attribute . . . 156

6.21.2 CreatureAdapter . . . 157
6.21.3 GameObject . . . 168
6.21.4 GameObjects . . . 171
6.21.5 IAttackable . . . 171
6.21.6 ManaCrystal . . . 172
6.21.7 Obstacle . . . 173
6.21.8 PlayerCharacter . . . 174
6.21.9 SummonableMinion . . . 175
6.22 Model.GameObject.Minions.Chaos . . . 177
6.22.1 HellhoundMinion . . . 177
6.22.2 HydraMinion . . . 178
6.22.3 ManticoreMinion . . . 178
6.22.4 SkeletonMinion . . . 179
6.22.5 VampireMinion . . . 180
6.22.6 ZombieMinion . . . 180
6.23 Model.GameObject.Minions.Law . . . 181
6.23.1 IcarusTowerMinion . . . 181
6.23.2 PaladinMinion . . . 182
6.23.3 PegasusMinion . . . 182
6.23.4 SapphireDragonMinion . . . 183
6.24 Model.GameObject.Minions.Neutral . . . 184
6.24.1 AirElementalMinion . . . 184
6.24.2 EagleMinion . . . 184
6.24.3 GiantMinion . . . 185
6.24.4 LionMinion . . . 186
6.25 Model.Map . . . 186
6.25.1 AxialTransform . . . 186
6.25.2 CubeTransform . . . 189
6.25.3 FloatCubeTransform . . . 190
6.25.4 Map . . . 191
6.25.5 MapGenArgs . . . 197
6.25.6 MapGenerationKeywords . . . 198
6.25.7 MapGenerator . . . 199
6.25.8 Pathfinding . . . 206
6.25.9 PerlinNoise . . . 211
6.25.10 Tile . . . 213
6.26 Resource . . . 217
6.26.1 Dimensions . . . 217
6.26.2 VideoOptions . . . 218
6.27 Resource.Constants.CardConstants . . . 218
6.27.1 CardConstants . . . 218
6.28 Resource.Constants.CardConstants.SpellConstants.Chaos . . . 219
6.28.1 VengeanceSpellConstants . . . 219

6.29 Resource.Constants.CardConstants.SpellConstants.Law 220
6.29.1 DecreeSpellConstants 220
6.29.2 DivineBowSpellConstants 221
6.29.3 DivineShieldSpellConstants 222
6.29.4 DivineSwordSpellConstants 223
6.30 Resource.Constants.CardConstants.SpellConstants.Neutral 224
6.30.1 MagicBoltSpellConstants 224
6.30.2 WindWalkerSpellConstants 225
6.31 Resource.Constants.CardConstants.SummonConstants 226
6.31.1 SummonConstants 226
6.32 Resource.Constants.CardConstants.SummonConstants.Chaos 227
6.32.1 HellhoundSummonConstants 227
6.32.2 HydraSummonConstants 228
6.32.3 ManticoreSummonConstants 229
6.32.4 SkeletonSummonConstants 229
6.32.5 VampireSummonConstants 230
6.32.6 ZombieSummonConstants 231
6.33 Resource.Constants.CardConstants.Law 232
6.33.1 IcarusTowerSummonConstants 232
6.33.2 PaladinSummonConstants 233
6.33.3 PegasusSummonConstants 233
6.33.4 SapphireDragonSummonConstants 234
6.34 Resource.Constants.CardConstants.Neutral 235
6.34.1 AirElementalSummonConstants 235
6.34.2 EagleSummonConstants 236
6.34.3 GiantSummonConstants 236
6.34.4 LionSummonConstants 237
6.35 Resource.Constants.CardConstants.WandConstants 238
6.35.1 AirWandConstants 238
6.35.2 CreatureWandConstants 239
6.35.3 RandomWandConstants 240
6.35.4 UndeadWandConstants 241
6.36 Resource.Constants.GeneralConstants 242
6.36.1 ActionConstants 242
6.36.2 ControllerConstants 242
6.36.3 GameConstants 244
6.36.4 GameObjectConstants 244
6.36.5 MapConstants 246
6.36.6 MenuConstants 249
6.36.7 NetworkConstants 253
6.36.8 PlayerConstants 254
6.36.9 PopUpConstants 255
6.36.10 SoundConstants 256
6.36.11 UIConstants 257

6.37 Resource.Constants.MinionConstants . . . 258
6.37.1 PlayerCharacterMinionConstants . . . 258
6.38 Resource.Constants.MinionConstants.Chaos . . . 259
6.38.1 HellhoundMinionConstants . . . 259
6.38.2 HydraMinionConstants . . . 260
6.38.3 ManticoreMinionConstants . . . 261
6.38.4 SkeletonMinionConstants . . . 262
6.38.5 VampireMinionConstants . . . 263
6.38.6 ZombieMinionConstants . . . 264
6.39 Resource.Constants.MinionConstants.Law . . . 265
6.39.1 IcarusTowerMinionConstants . . . 265
6.39.2 PaladinMinionConstants . . . 266
6.39.3 PegasusMinionConstants . . . 267
6.39.4 SapphireDragonMinionConstants . . . 268
6.40 Resource.Constants.MinionConstants.Neutral . . . 269
6.40.1 AirElementalMinionConstants . . . 269
6.40.2 EagleMinionConstants . . . 269
6.40.3 GiantMinionConstants . . . 270
6.40.4 LionMinionConstants . . . 271
6.41 Resource.Images . . . 272
6.41.1 Animations . . . 272
6.41.2 CardExclusiveImages . . . 273
6.41.3 GOImages . . . 274
6.41.4 HexImages . . . 276
6.41.5 Images . . . 277
6.41.6 UIElementsImages . . . 278
6.41.7 WandImages . . . 278
6.42 Util . . . 279
6.42.1 Stack . . . 279
6.43 View.Menus . . . 280
6.43.1 MainMenu . . . 280
6.43.2 MapOptionsMenu . . . 282
6.43.3 Menu . . . 285
6.43.4 MultiplayerMenu . . . 288
6.43.5 OptionsMenu . . . 291
6.43.6 PauseMenu . . . 292
6.43.7 SoundMenu . . . 294
6.44 View.PopUps . . . 297
6.44.1 ConnectionPopUp . . . 297
6.44.2 MapSavePopUp . . . 298
6.44.3 PopUp . . . 298
6.44.4 WinPopUp . . . 300
6.45 View.Scenes . . . 301
6.45.1 ChaosScene . . . 301

6.45.2 GameScene . . . 305
6.45.3 GameSetupScene . . . 309
6.45.4 GameSetupSceneMP . . . 314
6.45.5 MenuScene . . . 318
6.46 View.Stylesheets . . . 319
6.46.1 MenuButton . . . 319
6.46.2 MenuItem . . . 319
6.46.3 MenuSlider . . . 319
6.46.4 MenuTextField . . . 319
6.47 View.Tooltip . . . 320
6.47.1 CardTooltip . . . 320
6.47.2 GOTooltip . . . 322
6.47.3 MinionTooltip . . . 322
6.47.4 TileTooltip . . . 324
6.47.5 Tooltip . . . 325
6.47.6 TooltipBox . . . 329
6.47.7 WandTooltip . . . 331
6.48 View.UIElements . . . 332
6.48.1 CardView . . . 332
6.48.2 ChaosBackground . . . 336
6.48.3 ChaosMenuButton . . . 336
6.48.4 ChaosVBox . . . 339
6.48.5 DeckUI . . . 340
6.48.6 FluxBar . . . 342
6.48.7 HandView . . . 346
6.48.8 IChaosUI . . . 347
6.48.9 ManaBar . . . 347
6.48.10 MapView . . . 350
6.48.11 MusicAndEffect . . . 356
6.48.12 PlayerWandView . . . 359
6.48.13 RoundButton . . . 361
6.48.14 TextWithBackground . . . 365
6.48.15 TileView . . . 367
6.48.16 TurnPlayerDisplay . . . 368
6.48.17 WandView . . . 370

7 Art **372**
7.1 AirElemental . . . 372
7.2 AirWand . . . 373
7.3 BlueExplosion . . . 373
7.4 Buff . . . 373
7.5 CreatureWand . . . 374
7.6 Death . . . 374
7.7 Deck . . . 375

7.8 Decree . . . 375
7.9 DivineBow . . . 375
7.10 DivineShield . . . 375
7.11 DivineSword . . . 375
7.12 Eagle . . . 376
7.13 EndTurnHover . . . 376
7.14 EndTurnIdle . . . 376
7.15 EndTurnPressed . . . 376
7.16 Flux . . . 377
7.17 Frame_Player_Lime . . . 377
7.18 Frame_Player_Red . . . 377
7.19 Frame_Player_Yellow . . . 377
7.20 GameBackground . . . 378
7.21 Giant . . . 378
7.22 Hellhound . . . 378
7.23 Hex_Frame . . . 378
7.24 Hex_Frame_Highlighted_atk . . . 379
7.25 Hex_Frame_Highlighted_atk_move . . . 379
7.26 Hex_Frame_Highlighted_move . . . 379
7.27 Hex_Frame_Selected . . . 379
7.28 Hex_Height_0 . . . 379
7.29 Hex_Height_1 . . . 379
7.30 Hex_Height_2 . . . 380
7.31 Hex_Height_3 . . . 380
7.32 Hit . . . 380
7.33 Hydra . . . 380
7.34 IcarusTower . . . 380
7.35 MagicBolt . . . 380
7.36 ManaBar . . . 381
7.37 ManaCost . . . 381
7.38 Mana_Crystal . . . 381
7.39 Manticore . . . 381
7.40 MenuBackground0 . . . 382
7.41 MenuBackground1 . . . 383
7.42 MenuBackground2 . . . 383
7.43 MenuBackground3 . . . 384
7.44 NoImage . . . 384
7.45 Obstacle . . . 384
7.46 Paladin . . . 384
7.47 Pegasus . . . 385
7.48 PlayerCharacter_Wizard_lime . . . 385
7.49 PlayerCharacter_Wizard_purple . . . 385
7.50 PlayerCharacter_Wizard_red . . . 385
7.51 PlayerCharacter_Wizard_yellow . . . 385

7.52 RandomWand 386
7.53 SapphireDragon 386
7.54 Skeleton 386
7.55 Summon 387
7.56 Lion 387
7.57 UndeadWand 387
7.58 Vampire 388
7.59 Vengeance 388
7.60 Windwalker 388
7.61 Zombie 388

1 Aufgabenstellung

„Sie sollen die Mechanik (Law-Modus) des Spiels Chaos Reborn (http://www.chaos-reborn.com/) in Java nach den unten genannten Anforderungen implementieren und visualisieren, dabei sollen Sie kreativ sein. Beachten Sie, dass Sie nur die unten genannten Anforderungen umsetzen müssen und im 2D bleiben.“
Quelle: PROG2 SoSe 20 Aufgabe.pdf

2 Anforderungsdefinition

1. Spieler 1 spielt gegen Spieler 2 am selben PC oder über Netzwerk. Das Ziel ist die Auslöschung des anderen Spielers. Jeder Spieler hat einen Wert für Angriff, Verteidigung, Bewegungsradius, Mana und Lebenspunkte.

2. Das Spiel ist rundenbasiert. Es soll **keine** KI implementiert werden.

3. Es wird auf einer 2D Schachbrett-Karte gespielt. Für die Karten gibt es die Größen Klein, Mittel und Groß.

4. Auf der Karte können Hindernisse sein. Felder mit Hindernissen können nicht betreten werden.

5. Es gibt Manaquellen, die die Spieler beim Betreten des Feldes einmalig mit Mana versorgen. Die Positionen dieser Manaquellen und deren Anzahl wird zufällig bestimmt.

6. Es gibt Erhöhungen der Felder (bis 2 Ebenen). Manche Wesen können solche Erhöhungen nicht betreten. Visualisieren Sie diese Erhöhungen geeignet. So könnten diese mit Grauwerten dargestellt werden, wobei Schwarz ein Hindernis darstellt.

7. Jeder Spieler besitzt ein Deck von Karten. Karten können neutral, chaotisch oder rechtschaffen sein.

8. Mit dem Ausspielen einer Karte wird ein Wesen erschaffen oder ein Zauber ausgelöst.Das Ausspielen einer Karte kostet eine von der Karte bestimmte Menge Mana.

9. Übernehmen Sie die Wesen und Zauber aus Chaos Reborn oder denken Sie sich neue aus.

10. Die Spieler werden zu Beginn des Spiels mit einem zufälligen Zauberstab ausgerüstet. Diese Zauberstäbe bestimmen die Art des Decks, die Anzahl der Karten auf der Hand, den Verbrauch von Mana und wenn sie wollen noch mehr.

11. Es gibt eine Mana-Flux-Anzeige, die zu Beginn des Spiels neutral ist. Die Anzeige verändert sich in die entsprechende Richtung der ausgespielten Karte. Wird eine

chaotische Karte gespielt, dann wird die Anzeige in Richtung chaotisch verändert und damit kosten chaotische Karten weniger Mana. Überlegen Sie sich hier eine gute Balance, die auch von Zauberstäben verändert werden kann.

12. Spieler und Wesen können sich pro Runde um eine bestimmte Anzahl von Feldern bewegen und angreifen. Manche Wesen haben Spezialangriffe. Denken Sie bei der Bewegung an Hindernisse und Erhöhungen. Manche Wesen können fliegen.

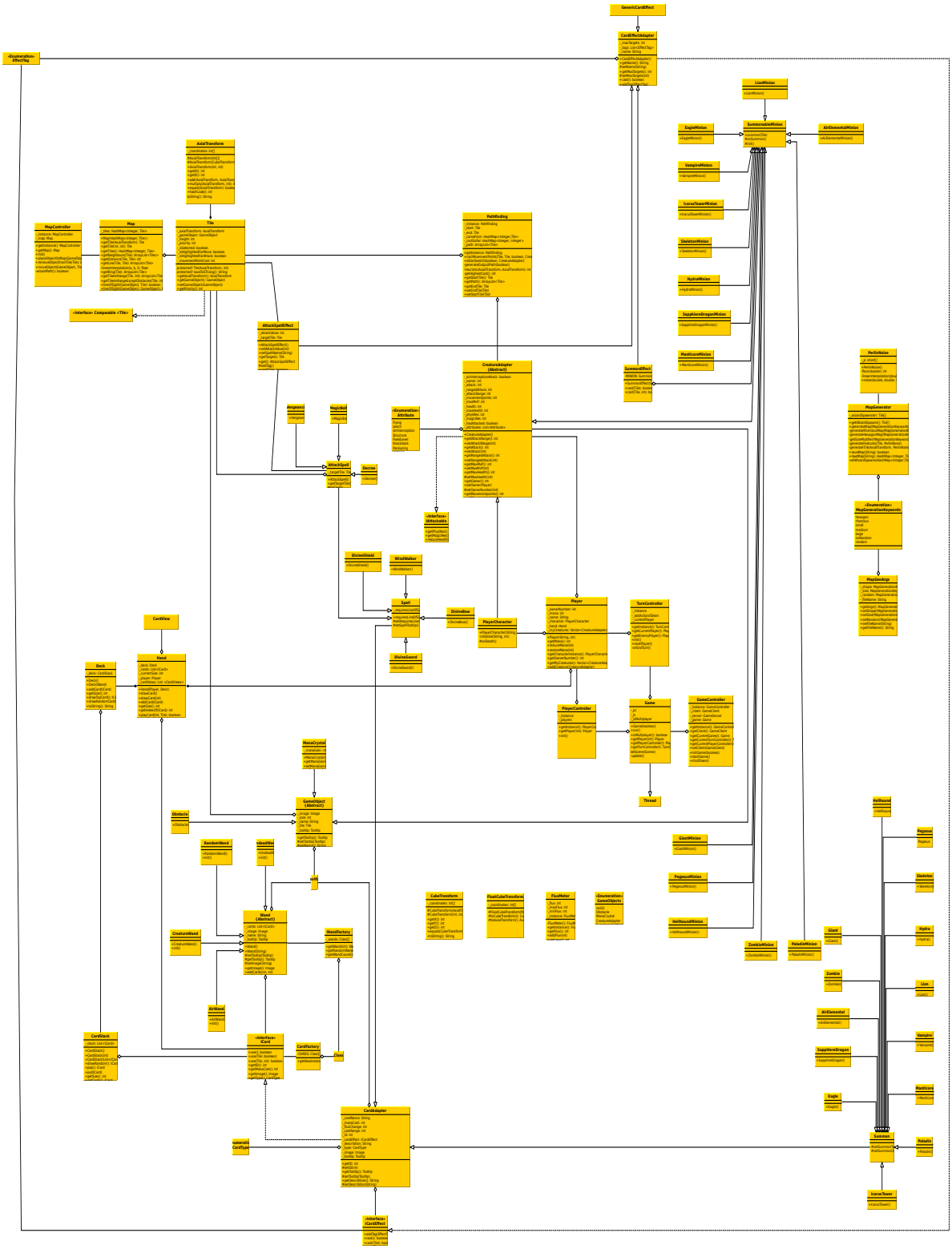
13. Wesen haben zumindest Lebenspunkte, Angriffswert, Verteidigungswert und Bewegungsradius.

14. Die Spieler und die Wesen werden mit 2D-Bildern auf der Karte dargestellt.

15. Setzen Sie Sound geeignet ein und visualisieren Sie die auf der Karte stattfindenden Aktionen (Selektion, Angriff, Bewegung, usw.) geeignet.

3 Entwurf

Um die Anforderungen umzusetzen und das Spiel zu implementieren haben wir einen Entwurf erarbeitet indem wir uns bei einem Brainstorming überlegt haben welche Klassen wir brauchen, welche Attribute diese benötigen und ob wir mit Quadraten oder Hexagonen arbeiten möchten. Wir haben uns für Hexagone entschieden und aus unseren Ergebnissen vom Brainstorming ein UML-Diagramm erstellt dass wir laufend überarbeitet haben.
Das Diagramm enthält nur den Teil des Quellcodes in dem wir das Spiel und seine Anforderungen modellieren und ist zoombar ohne unleserlich zu werden.


GenericCardEffect
CardEffectAdapter
LionMotion
SummonableMotion
EagleMotion
AirElementalMotion
VampireMotion
IcarusTowerMotion
SkeletonMotion
HydraMotion
SapphireDragonMotion
ManticoreMotion
PerlinNoise
MapGenerator
MapController
Map
Tile
AxialTransform
PathFinding
CreatureAdapter (Abstract)
SummonEffect
AttackSpellEffect
AttackSpell
Decree
Spell
DivineBlow
PlayerCharacter
Player
TurnController
DivineShield
WindWalker
DivineSword
CardView
Hand
Deck
Game
GameController
PlayerController
Thread
MagicCrystal
GameObject (Abstract)
Obstacle
RandomWord
AirWand
Card
Minion (Abstract)
CreatureWand
MinionFactory
CardStack
CardFactory
ICard
CardAdapter
IAttributes
GiantMotion
PegasusMotion
HellhoundMotion
ZombieMotion
PaladinMotion
Giant
Zombie
AirElemental
SapphireDragon
Eagle
Paladin
IcarusTower
Pegasus
Skeleton
Hydra
Lion
Vampire
Manticore
Hellhound
CubeTransform
FloatCubeTransform
FluxMeter
Summon

4 Benutzungshinweise

4.1 Das Hauptmenü

Das Hauptmenü ist sichtbar, sobald das Spiel gestartet ist. Es stellt folgende Möglichkeiten bereit:

Ein lokales Spiel starten Die Schaltfläche „Start Local“startet ein Spiel, bei dem beide Spieler an demselben Rechner gegeneinander spielen. Für weitere Informationen siehe Abschnitt „Ein lokales Spiel starten “.

Ein Spiel über Netzwerk starten Die Schaltfläche „Start Network“ startet ein Spiel, bei dem beide Spieler über Netzwerk gegeneinander spielen. Für weitere Informationen siehe Abschnitt „Ein Netzwerkspiel Starten“.

Optionen Die Schaltfläche „Options“ ermöglicht zugriff auf diverse Einstellungen. Für weitere Informationen siehe Abschnitt „Die Optionen“.

Beenden Die Schaltfläche „Exit“ beendet das Spiel.

4.2 Die Optionen

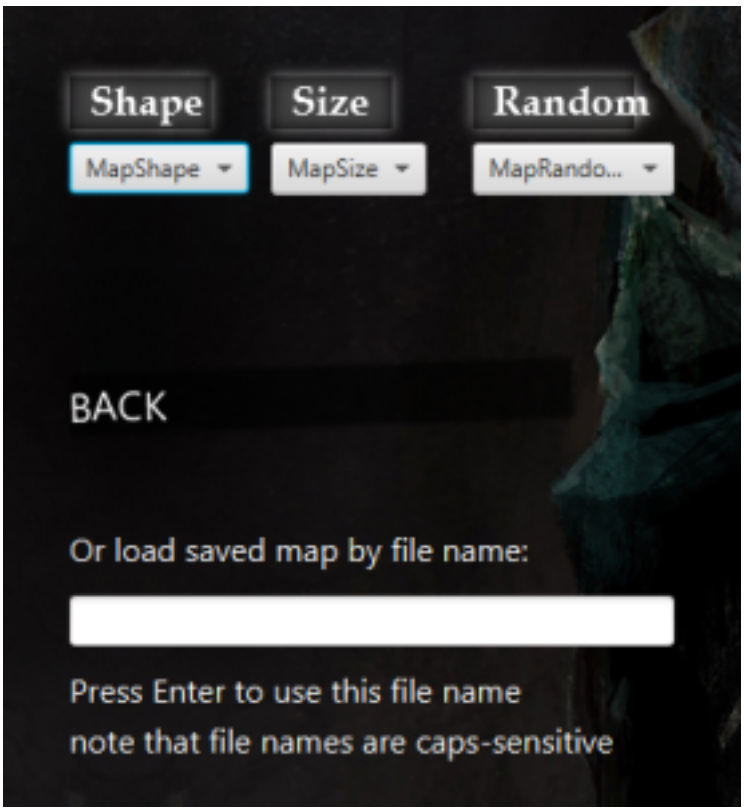
Karteneinstellungen Unter „Map Options“ kann die Karte ausgewählt werden, auf der gespielt werden soll.
Für weitere Informationen siehe Abschnitt „Die Karteneinstellungen“ .

Audioeinstellungen Unter „Sound“ kann die Ambientemusik aktiviert oder deaktiviert, sowie ihre Lautstärke angepasst werden.

Zurück zum Hauptmenü Ein Druck auf die „Back“ Schaltfläche bringt Sie zurück ins Hauptmenü.

4.2.1 Die Karteneinstellungen

Hier können unter drei Überschriften mittels Drop-Down-Menüs Optionen für die Karte ausgewählt werden.



Werden hier keine Veränderungen vorgenommen wird eine unter „Hexagonal, Medium, Not Random“ gespeicherte Karte als Standardkarte ausgewählt. Sollten Veränderungen vorgenommen werden, sollten Sie dies unter allen drei Überschriften tun. Alternativ kann der Dateiname einer zuvor gespeicherten Karte eingegeben werden, um diese Karte zu verwenden. Zum Speichern von Karten finden Sie weitere Informationen unter „Das Pausemenü“ .

Kartenform Unter „Shape“ Kann zwischen einer rhombusförmigen und sechseckförmigen Karte gewählt werden.

Kartengröße Unter „Size“ kann zwischen 3 unterschiedlichen Größen für Karten gewählt werden.

Zufall Unter „Random“ kann zwischen „Random“ - einer zufällig generierten Karte - und „Not Random“ - einer per Hand erstellten Karte gewählt werden. Nicht zufällige Karten sind in allen drei Größen, jedoch nur für sechseckige Karten verfügbar.

Namenseingabe Haben Sie zuvor eine Karte abgespeichert, können sie auch im sichtbaren Textfeld deren Dateinamen (ohne Endung .map) eingeben und mittels Druck auf die Enter-Taste ihrer Tastatur bestätigen. Die entsprechende Karte wird beim Start des Spiels geladen werden. Sollten Sie sich umentscheiden und doch keine Karte laden wollen, entfernen Sie jeglichen Text aus dem Eingabefeld und bestätigen erneut mittels Enter.

Haben Sie alle gewünschten Einstellungen vorgenommen, gelangen Sie mittels Druck auf die „Back“ Schaltfläche zurück.

4.3 Das Pausemenü

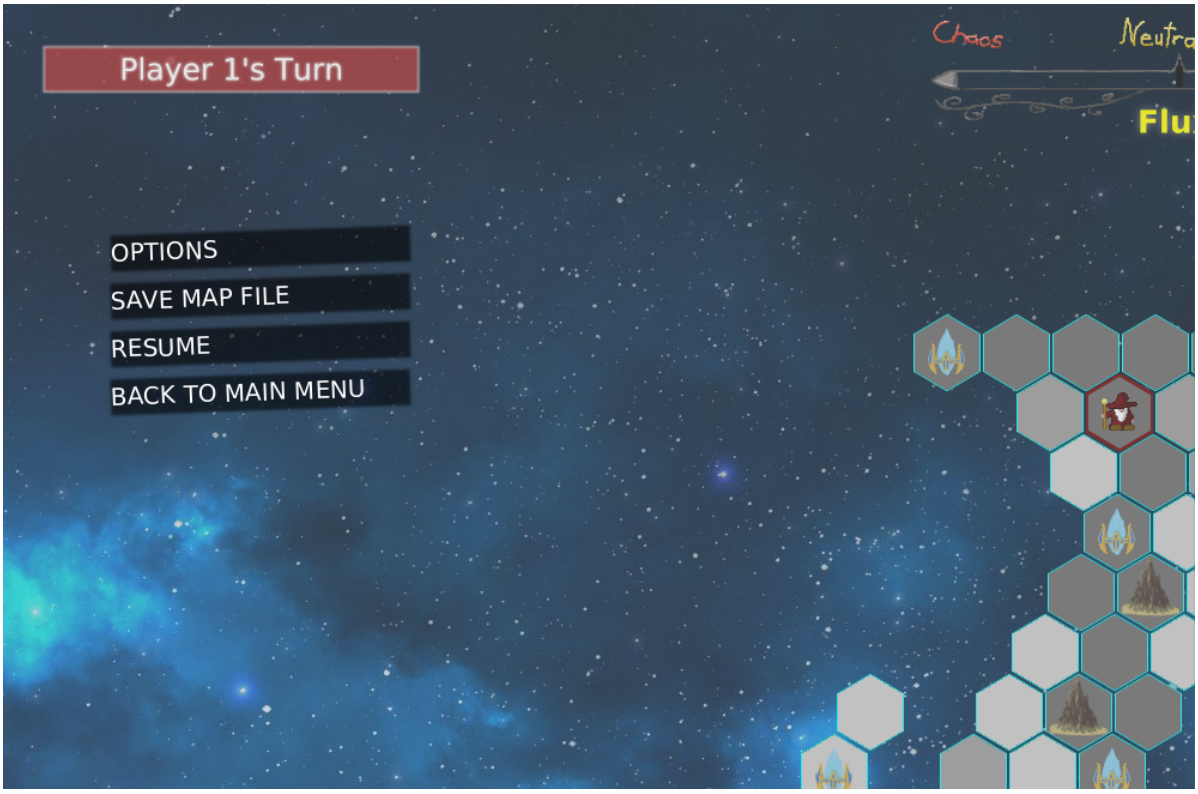
Das Pausemenü kann während eines Laufenden Spiels mittels Druck auf die Escape-Taste Ihrer Tastatur geöffnet werden. Ihnen stehen hier folgende Möglichkeiten bereit:

Optionen Ähnllich wie im Hauptmenü (siehe „Das Hauptmenü“) können Sie unter „Optionen“ Einstellungen vornehmen. Beachten Sie, dass Änderungen der Karteneinstellungen während des laufenden Spieles keine Auswirkung haben.

Eine Karte Speichern Mit Auswahl der „Save Map File“ Schaltfläche öffnet sich ein neues Fenster mit dem Titel „Saving Map“ . In einem Textfeld können Sie einen Dateinamen angeben, unter dem Sie die Karte abspeichern können. Ein Klick auf bestätigen schließt das Speichern ab. Beachten Sie, dass dies lediglich die Karte selbst, nicht den aktuellen Stand ihres Spieles sichert. Möchten Sie die Karte doch nicht abspeichern schließen sie dieses Fenster mit einem Klick auf das „X“ in der oberen rechten Ecke.

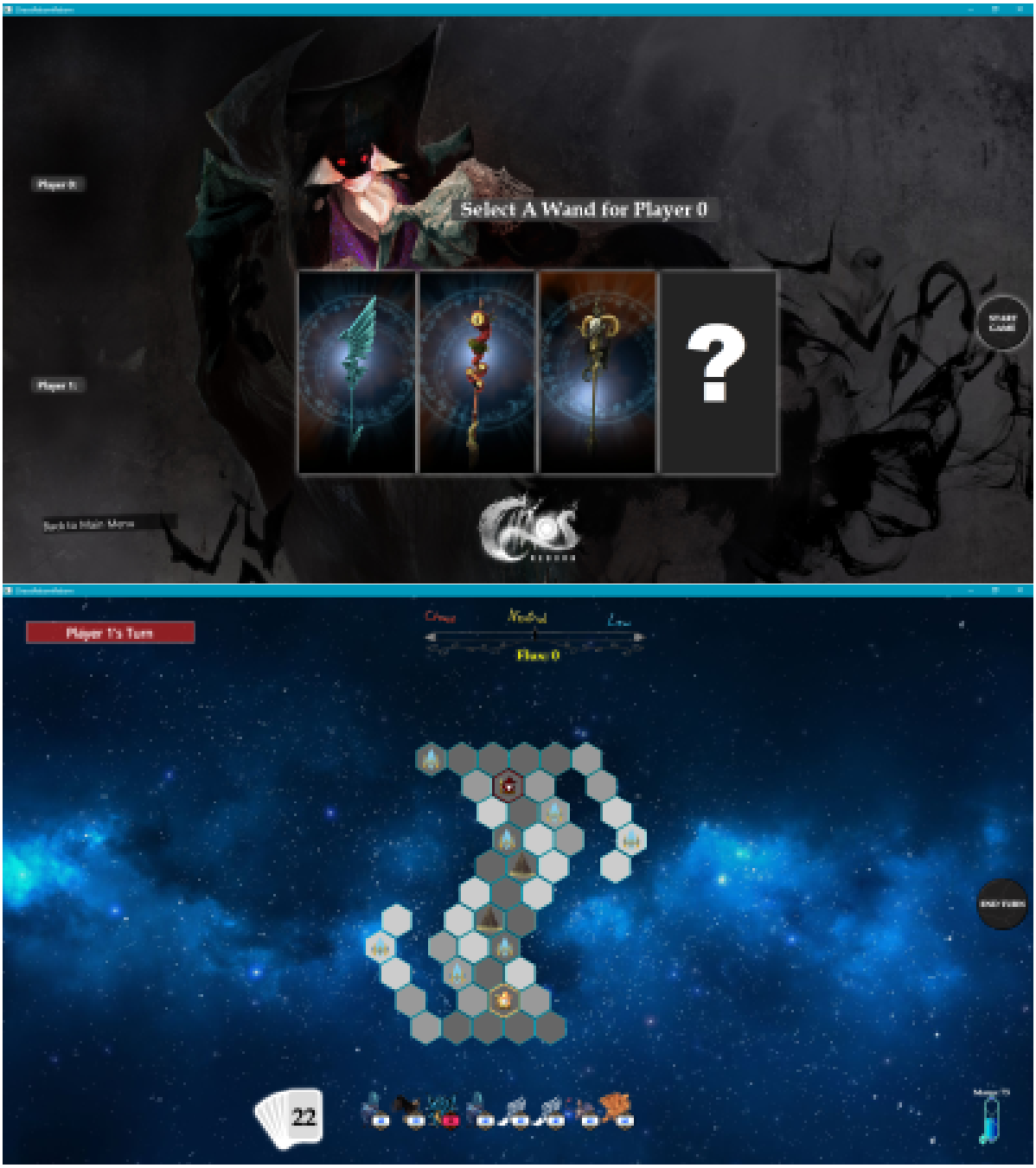
Weiterspielen Ein Klick auf „Resume“ setzt das laufende Spiel fort.

Zum Hauptmenü Ein Klick auf „Back to Main Menu“ bringt Sie zurück ins Hauptmenü. Dabei geht ihr aktueller Spielstand verloren.



4.4 Ein lokales Spiel starten

Im lokalen Spiel spielen beide Kontrahenten an einem Rechner und wechseln zwischen Ihren Zügen die Kontrolle über Maus und Tastatur ab. Nachdem Sie im Hauptmenü auf die Schaltfläche „Start Local“ geklickt haben, gelangen Sie zunächst in ein Auswahlmenü für den Zauberstab beider Spieler.



Beide Spieler wählen hier den Zauberstab aus, mit dem Sie das Spiel bestreiten möchten. In der oberen rechten Ecke des Bildschirmes wird dabei angezeigt welche Spielkarten wie oft im Deck eines Stabes enthalten sind. Die „Random Wand“ auszuwählen weist dem Spieler erst zu Spielbeginn einen zufälligen Zauberstab zu. Mehr Informationen zu Zauberstäben und Spielkarten finden Sie unter „Die Spielmechanik“. Sobald beide Spieler mit ihrer Wahl zufrieden sind klicken Sie auf „Start Game“. Wie gespielt wird erfahren Sie unter „Das Spiel“. Zu jedem Zeitpunkt der Vorbereitung gelangen Sie durch einen Klick auf „Back to Main Menu“ zurück ins Hauptmenü.

4.5 Ein Netzwerkspiel starten

Möglicherweise sind für das Spiel über Netzwerk Portfreigaben in ihrem Router und/oder ihrer Firewall nötig. Das Spiel verwendet den Port 6066.
Um ein Netzwerkspiel zu starten, öffnen Sie zunächst das Spiel auf zwei Rechnern oder starten das Spiel auf demselben Rechner zweimal.
Zunächst klicken Sie in einem Spiel auf „Host“. Ihnen werden nun Verbindungsinformationen angezeigt, mittels derer ein zweiter Spieler Ihrem Spiel beitreten kann. Im anderen Spiel klicken sie auf die Schaltfläche „Join“, um einem Spiel beizutreten. Es erscheint ein Textfeld zur Eingabe der IP-Adresse des Hosts. Sind beide Spiele auf demselben Rechner gestartet geben Sie hier „localhost“ ein. Andernfalls geben Sie die IP-Adresse des Hosts ein. Bestätigen Sie die Eingabe mittels Druck auf die Enter-Taste oder „Join Game!“ Schaltfläche. Wurde die Verbindung erfolgreich hergestellt, gelangen beide Spieler in ein Auswahlmenü für Ihren Zauberstab. Hat ein Spieler seine Auswahl getroffen klickt er auf „Ready“. Haben beide Spieler dies getan, beginnt das Spiel. Wie gespielt wird erfahren Sie unter „Das Spiel“. Falls sie eine eigens erstellte Map bespielen wollen muss diese von beiden Spielern geladen werden.

4.6 Das Spiel

(1) Zugspieler-Anzeige Hier sehen Sie, welcher Spieler derzeit am Zug ist.

(2) Mana-Flux-Anzeige Hier sehen Sie, den aktuellen Stand des Flux; ob es gerade in Richtung Chaos oder Rechtschaffenheit verschoben oder neutral ist.

(3) Das Spielfeld Das Spielfeld zeigt ihnen die Karte, auf der Sie spielen, sowie die Positionen von Zauberern, Kreaturen, Hindernissen und Manakristallen. Sie können einzelne Felder (Sechsecke) mit der linken Maustaste anklicken, um sie Auszuwählen. Dies hebt dieses Feld grün hervor. Sie können die Position des Spielfeldes auf dem Bildschirm verändern indem sie es per drag-and-drop über den Bildschirm ziehen.

(4) Das Deck Hier können Sie sehen, wie viele Karten noch in Ihrem deck vorhanden sind.

(5) Die Hand Die Hand zeigt Ihnen, welche Karten sie gezogen haben, wie viel Mana diese kosten (blaue Zahl) und ob sie diese gerade ausspielen können (haben Sie zu wenig Mana werden die Kosten einer Karte rot markiert). Außerdem wird ihnen, falls das Flux nicht neutral ist, die Kosten welcher Karten derzeit durch das Flux beeinflusst werden.

(6) Tooltips In der rechten, oberen Ecke werden Ihnen Tips zu dem Feld oder Zauber angezeigt, das oder den Sie derzeit ausgewählt haben. Befindet sich auf dem ausgewählten Feld eine Kreatur, werden auch zu ihr hier Informationen angezeigt.

(7) Zug-Beenden-Schaltfläche Ein klick auf „End Turn“ beendet Ihren Zug. Nun ist ihr Kontrahent an der Reihe.

(8) Mana-Anzeige Hier sehen Sie, wie viel Mana Sie derzeit zur Verfügung haben, um Zauber auszuspielen oder Kreaturen zu beschwören.

4.6.1 Eine Bewegung ausführen

Um eine Kreatur, oder Ihren Zauberer zu bewegen, wählen Sie zunächst mit der linken Maustaste das Feld auf dem sich die Kreatur befindet aus. Daraufhin werden ihnen in hellgrün alle Felder hervorgehoben, zu denen sich diese Kreatur bewegen kann. Mit einem Rechtsklick auf eines dieser Felder bewegen Sie die Kreatur dorthin.

4.6.2 Einen Manakristall einsammeln

Um einen Kristall einzusammeln, bewegen Sie eine Ihrer Kreaturen auf das Feld, auf dem sich der Kristall befindet. Der Kristall verschwindet und Ihnen wird Mana gutgeschrieben. Eine Bewegung, die lediglich über einen Kristall hinweg verläuft sammelt diesen nicht ein.

4.6.3 Einen Angriff ausführen

Um eine feindliche Kreatur anzugreifen wählen Sie zunächst wieder mit Linksklick ein Feld, auf dem sich eine Ihrer eigenen Kreaturen befindet. Nun bewegen Sie Ihre Kreatur auf ein Feld, von dem aus es den Feind angreifen kann. Wählen Sie erneut das Feld, ihrer eigenen Kreatur aus. Jetzt wird ihnen das Feld, auf dem sich der Feind befindet rot

hervorgehoben. Rechtsklicken Sie auf dieses um den Angriff auszuführen. Alternativ können Sie auch, wenn bereits Felder mit Gegnerischen Kreaturen hervorgehoben werden, nachdem Sie zum ersten Mal Ihre Kreatur wählen, sofort auf den Gegner Rechtsklicken. Ihre Kreatur wird sich nun so weit bewegen wie nötig, um den Angriff durchzuführen.

4.6.4 Eine Spielkarte verwenden

Linksklicken sie zunächst auf auf die Karte in Ihrer Hand. Es werden ihnen mögliche Zielfelder in rot hervorgehoben. Linksklicken Sie eines dieser Felder, um die Karte auszuspielen.

4.7 Die Spielmechanik

4.7.1 Spielziel

Ziel des Spieles ist es den feindlichen Zauberer zu besiegen, indem seine Lebenspunkte auf 0 reduziert werden.

4.7.2 Zauberstäbe

4.7.3 Die Hand & Das Deck

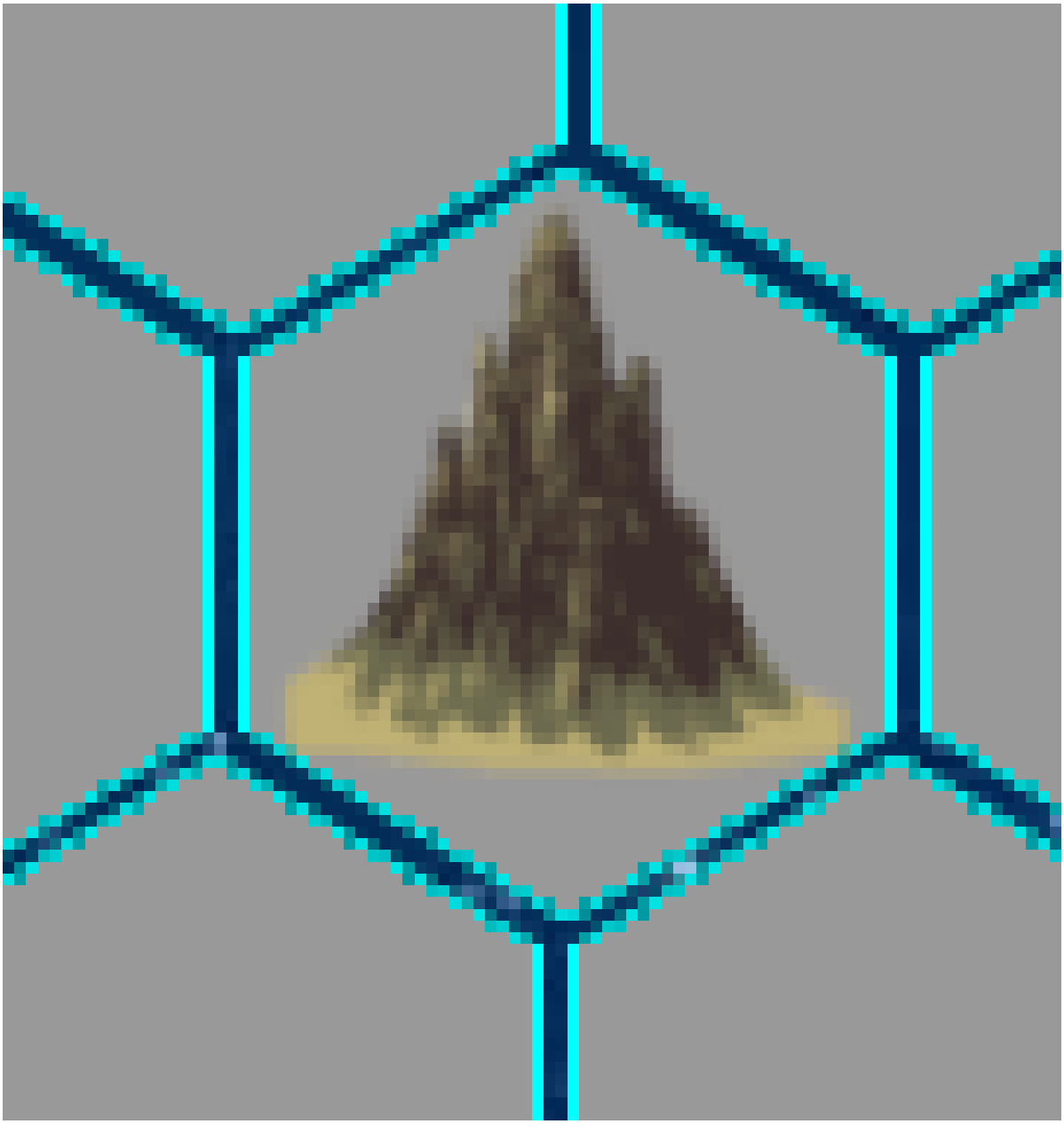
Der Zauberstab bestimmt, welche Spielkarten wie oft im Deck des Spielers vorkommen. Das Deck enthält diese Karten in zufälliger Reihenfolge. Die Hand besteht aus Karten, die vom Deck gezogen werden. Zu Beginn jedes Zuges wird eine neue Karte gezogen. Ist das Deck leer, bleiben nur noch die Karten in der Hand. Ist auch diese leer bleibt nur zu hoffen, dass die Kreaturen auf dem Feld zum Sieg reichen...

4.7.4 Der eigene Zauberer

Der Zauberer selbst hat dieselben Werte und Attribute wie andere Kreaturen auch (siehe Abschnitt „Kreaturen“). Sinken die Lebenspunkte des Zauberers auf oder unter 0 ist das Spiel verloren.

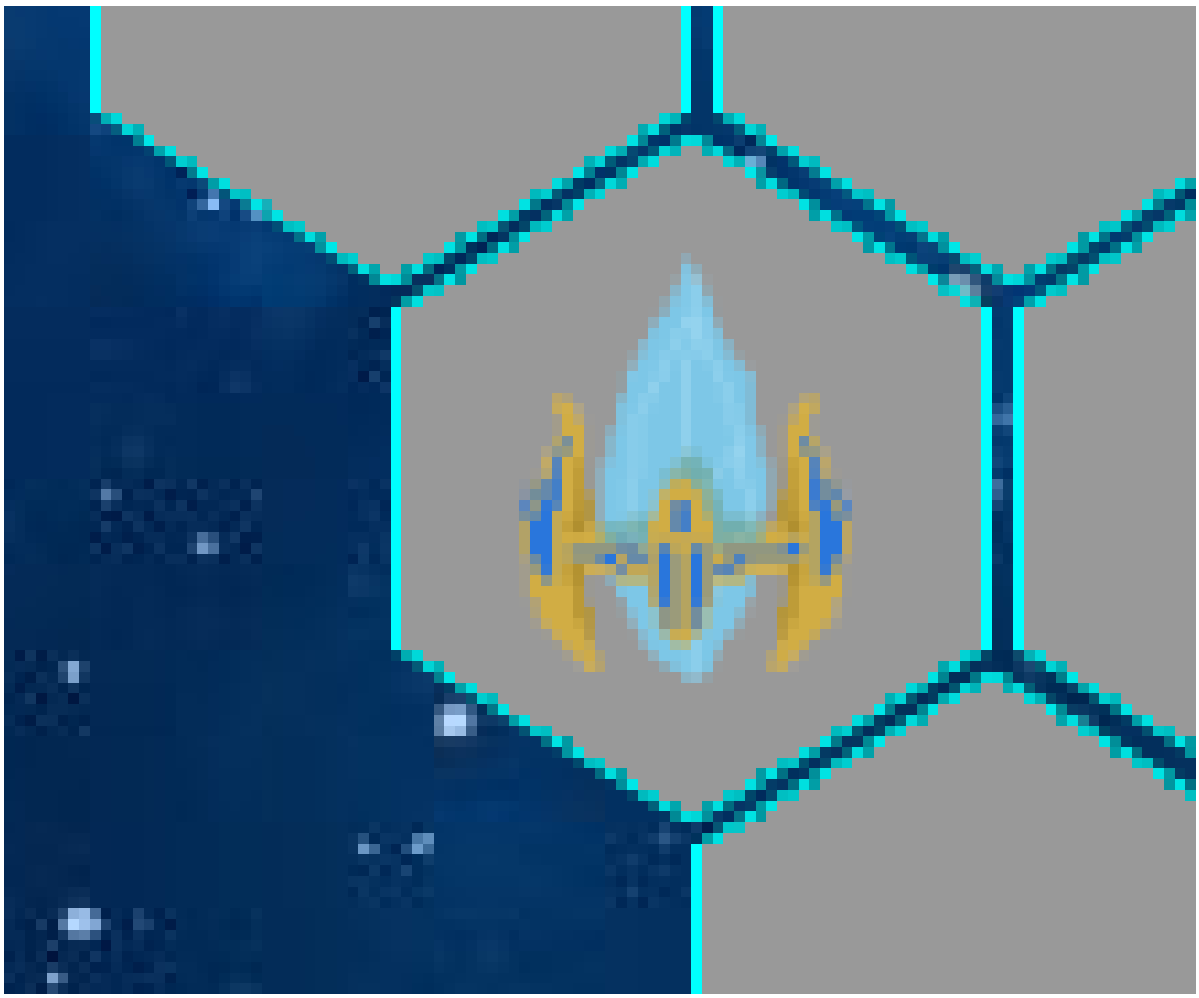
4.7.5 Angirffe & Bewegungen

Bewegung Kreaturen können sich abhängig von ihren Bewegungspunkten unterschiedlich weit über die Karte bewegen. Bewegungen werden durch die Gegebenheiten der Karte eingeschränkt. So können sich Kreaturen nicht über Hindernisse hinweg bzw. durch sie hindurch bewegen.



Außerdem haben Felder der Karte unterschiedliche Höhen. Ist der unterschied zweier aneinander grenzender Felder größer als 1, kann die Höhendifferenz von einer Kreatur nicht überwunden werden (es gibt Ausnahmen, siehe „Attribute“).

Sich auf einen Manakristall zu bewegen sammelt diesen für den Spieler ein (siehe „Mana & Flux“).



Angriff Es wird unterschieden zwischen Nah- und Fernkampfangriffen. Ein Nahkampfangriff kann gegen eine angrenzende Kreatur ausgeführt werden. Nahkampfangriffe fügen sowohl dem Angreifer, als auch dem Verteidiger schaden zu. Bei Fernkampfangriffen erleidet nur der Verteidiger Schaden. Um einen Fernkampfangriff auszuführen, muss der Angreifer das Ziel sehen können. Das bedeutet, dass sich zwischen ihm und dem Ziel keine Hindernisse befinden und dass zwischen ihnen keine Felder höher gelegen sind als beide Kreaturen groß sind.

4.7.6 Zauberei

Kreaturen beschwören beschworene Kreaturen sind austauschbare Kämpfer, deren Verlust nicht zum Verlieren des Spiels führt. Sie können sich bewegen, angreifen und angegriffen werden.

4.7.7 Zaupersprüche

Zaubersprüche unterstützen den Spieler, indem Sie eine Kreatur verletzen, den eigenen Zauberer stärken oder den Gegner Karten abwerfen lassen. Magischer Schlag („Magic Bolt“) Der Magische Schlag verursacht Schaden an der Zielkreatur.

Rache („Vengeance “) Rache verursacht Schaden an der Zielkreatur. Handelt es sich dabei um den gegnerischen Zauberer wirft der Gegner stattdessen 2 Karten aus seiner Hand ab.

Erlass („Decree“) Erlass verursacht Schaden an der Zielkreatur. Handelt es sich dabei um den gegnerischen Zauberer wirft der Gegner stattdessen 2 Karten aus seiner Hand ab.

Windläufer („Windwalker“) Windläufer erhöht die maximalen Bewegungspunkte des Zauberers um 1 und verleiht ihm das Attribut Fliegend. Wird der Zauber ein zweites Mal gespielt erhält der Zauberer den Bewegungspunkt, jedoch keine weiteren Attribute.

Göttlicher Schild („Divine Shield“) Göttlicher Schild erhöht die Physische Verteidigung des Zauberers.

Göttlicher Bogen („Divine Bow“) Göttlicher Bogen verleiht dem Zauberer einen Fernkampfangriff mit einer Reichweite von 3 Wird der Zauber ein zweites Mal gespielt steigt der Fernkampfangriffswert, jedoch nicht die Reichweite.

Göttliches Schwert („Divine Sword“) Göttliches Schwert erhöht den Angriffswert des Zauberers.

4.7.8 Kreaturen

Werte :

Lebenspunkte („Health“) Lebenspunkte geben an, wie verletzt eine Kreatur ist. Sinken sie auf oder unter 0, ist die Kreatur besiegt und wird vom Spielfeld entfernt.

Größe („Size“) Die Größe einer Kreatur wird verwendet um zu bestimmen, ob sie im Fernkampf angegriffen werden oder angreifen kann; genauer: ob sie als Ziel gesehen werden oder ihr Ziel sehen kann.

Angriff („Attack“) Der Anriffswert einer Kreatur bestimmt, wie viel Schaden sie im Nahkampf verursacht, h.h. wie viele Lebenspunkte der Gegner verliert, wenn sie kämpft.

Angriffsreichweite („Attack Range“) Die Angriffsreichweite gibt an, wie weit das Ziel eines Angriffes von der Kreatur entfernt sein kann. Eine 1 bedeutet, dass die Kreatur nur Feinde auf angrenzenden Feldern angreifen kann.

Fernkampfangriff („Ranged Attack“) Hat eine Kreatur eine Angriffsreichweite größer als 1, kann sie einen Fernkampfangriff durchführen. Hierbei wird zur Berechnung des Schadens der Fernkampfangriff, statt des Angriffs verwendet.

Physische Verteidigung („Physical Resistance“) Die Physische Verteidigung verringert den Schaden, den eine Kreatur durch einen Nah- oder Fernkampfangriff erleidet.

Magische Verteidigung („Magical Resistance“) Die Magische Verteidigung verringert den Schaden, den eine Kreatur durch Zaubersprüche erleidet.

Bewegungspunkte („Movement Points“) Die Bewegungspunkte geben an, wie weit eine Kreatur sich bewegen kann. Von einem Feld auf ein angrenzendes zu wechseln kostet einen Punkt, weitere Bewegungen entsprechend mehrere Punkte.

4.7.9 Attribute

Fliegend („Flying“) Fliegende Kreaturen können bei Bewegungen Höhenunterschiede größer als 1 überwinden.

Blutsauger („Leech“) Kreaturen mit Blutsauger heilen bei einem Nahkampfangriff schaden, den sie dem Ziel zufügen.

Luftabfangen („Air Interception“) Kreaturen mit diesem Attribut können einmal pro gegnerischem Zug einen zusätzlichen Angriff gegen eine Fliegende Kreatur ausführen, die eine Bewegung innerhalb ihrer Reichweite beginnt.

Struktur („Structure“) Strukturen sind gegen Zaubersprüche immun.

Feste Höhe („Fixed Level“) Die Kreatur kann keine Höhenunterschiede überwinden.

Rückstoß („Knockback“) Die Kreatur „schubst"das Ziel eines Angriffes um ein Feld zurück, falls dieses frei ist und das Ziel den Angriff überlebt.

Paralysierend („Paralyzing“) Führt eine Kreatur mit Paralysierend einen Fernkampfangriff aus, erhält das Ziel für eine Runde das Attribut Paralysiert.

Paralysiert („Paralyzed“) Eine Paralysierte Kreatur kann sich nicht bewegen und keine Angriffe ausführen.

Untot („Undead“) Eine Untote Kreatur kann nur durch andere Untote Kreaturen oder solche mit dem Attribut Untotschlächter verletzt werden.

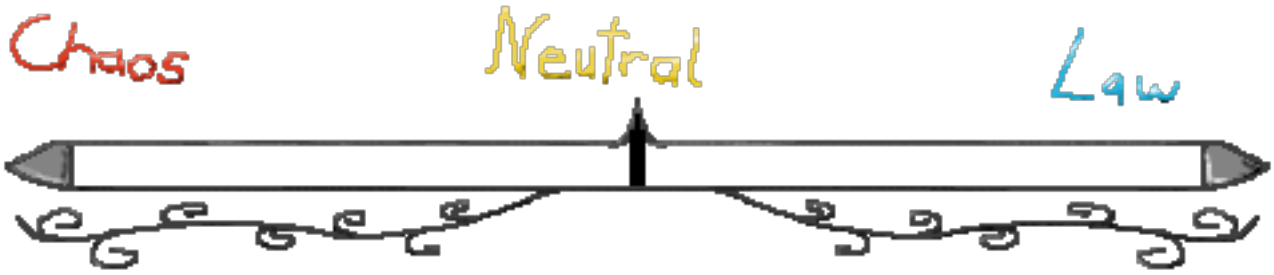
Untotschlächter („Undead Slayer“) Die Kreatur ist selber nicht Untot, kann jedoch Untote Kreaturen verletzen.

Einzelschuss („One Time Shot“) Die Kreatur kann, solange sie auf dem Spielfeld ist, einen einzigen Fernkampfangriff ausführen, statt beliebig viele.

4.7.10 Mana & Flux

Mana Kreaturen zu beschwören oder Zaubersprüche auszuspielen benötigt Mana. Die kosten hängen von der Jeweiligen Spielkarte und dem Aktuellen Flux (siehe „Flux“) ab. Jeder Spieler erhält zu Beginn seines Zuges Mana. Außerdem kann Mana durch das einsammeln von Manakristallen erhalten werden.

Flux Jede Spielkarte ist entweder rechtschaffen, neutral oder chaotisch. Eine rechtschaffene oder chaotische Karte auszuspielen verschiebt das aktuelle Flux in die entsprechende Richtung. Ist das Flux chaotisch, sind chaotische Karten für geringere Manakosten einsetzbar. Ist das Flux rechtschaffen, sind diese Karten günstiger. Neutrale Karten kosten immer gleich viel Mana und haben keinen Einfluss auf das Flux.



5 Anwendungsbeispiel

Nach dem man, wie in den Benutzungshinweisen beschrieben, ein lokales Spiel gestartet und die Zauberstäbe ausgewählt hat landet man im Spiel.

Es ist Spieler 1 am Zug, das ist der rot gekleidete Zauberer. Der Zauberer wird bewegt, zum Beispiel auf eines der Felder mit Mana das er dann einsammelt.

Weil kein Gegner in der Nähe ist kann der Zauberer nicht angreifen, er spielt ein paar Karten aus und platziert damit eigene Kreaturen um sich herum um.

Mit einem Klick auf den „END TURN“Button wird die Runde von Spieler 1 beendet und Spieler 2 ist am Zug. Spieler 2 tut es Spieler 1 gleich, bewegt sich und beschwört Kreaturen zur Hilfe gegen den Gegner.

Spieler 1 bewegt daraufhin seine Kreaturen in Richtung des Gegners und greift mit dem eigenen den gegnerischen Löwen an.

Weitere Kreaturen werden von den Zauberern beschworen und kämpfen gegeneinander.

Nach langem Kämpfen der beiden Zauberer hat Spieler 1 Spieler 2 besiegt

6 Quelldateien

6.1 Control

6.1.1 Main.java

Listing 1: Main.java

```java
package Control;

import Control.Controllers.SceneController;
import javafx.application.Application;
import javafx.stage.Stage;

/**
 * Main Class to start the application.
 */
public class Main extends Application
{

    /**
     * The start method starts the GUI of this application.
     *
     * @param primaryStage the Window out game is working on.
     */
    @Override
    public void start (Stage primaryStage)
    {
        primaryStage = SceneController.getInstance().getPrimaryStage();
        primaryStage.show();
    }

    /**
     * The main method starts to launch the application.
     *
     * @param args the command line arguments
     */
    public static void main (String[] args)
    {
        launch(args);
    }

}
```

6.1.2 Game.java

Listing 2: Game.java

```java
package Control;

import Control.Controllers.PlayerController;
import Control.Controllers.TurnController;
import Model.Player;
import static Resource.Constants.GeneralConstants.GameConstants.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * The Game holds all controllers, which are relevant for each individual game
 *
 * Extends from Thread, to enable each Game to run seperately if that is
 * required for out purposes.
 */
```

```
16 public class Game extends Thread
17 {

19     /**
20      * PlayerController for this Game.
21      */
22     private PlayerController _pc = null;

24     /**
25      * TurnController for this Game.
26      */
27     private TurnController _tc = null;

29     /**
30      * Indicates if the game is a multiplayer game or not.
31      */
32     private boolean _isMultiplayer = false;

34     /**
35      * Only sets the Name of the thread.
36      *
37      * @param multiplayer If this Game is played over a network, this must be
38      *                    true.
39      */
40     public Game (boolean multiplayer)
41     {
42         this.setName(GAME_THREAD_NAME);
43         this._isMultiplayer = multiplayer;
44     }

46     /**
47      * Initializes the controllers for this individual game.
48      */
49     @Override
50     public void run ()
51     {
52         this._pc = PlayerController.getInstance();
53         this._tc = TurnController.getInstance();

55         this._pc.init();
56         this._tc.init();

58         while (!this.isInterrupted())
59           {
60             try
61               {
62                 this.sleep(GAME_THREAD_SLEEP);
63                 this.update();
64               }
65             catch (InterruptedException ex)
66               {
67                 Logger.getLogger(Game.class.getName())
68                                                 .log(Level.SEVERE, null, ex);
69               }
70           }
71         System.err.println(this.getName() + GAME_THREAD_INTERRUPTED);
72     }

74     /**
75      *
76      */
77     private void update ()
78     {
79         //bleibt erstmal noch drinne
80     }
```

```
    /**
     * Getter to find out if the Game is a Multiplayer Game.
     *
     * @return - true if it is a Multiplayer Game.
     */
    public boolean isMultiplayer ()
    {
        return this._isMultiplayer;
    }

    /**
     * Function, to find a Player within a game.
     *
     * @param ownerNumber either 0 or 1, as there can only be 2 players present
     *                    in one game.
     *
     * @return returns the Player, corresponding to their owner number.
     */
    public Player getPlayer (int ownerNumber)
    {
        return this._pc.getPlayer(ownerNumber);
    }

    /**
     * Getter for the individual Player Controller.
     *
     * @return this game's Player Controller.
     */
    public PlayerController getPlayerController ()
    {
        return this._pc;
    }

    /**
     * Getter for the individual Turn Controller.
     *
     * @return this game's Turn Controller.
     */
    public TurnController getTurnController ()
    {
        return this._tc;
    }

}
```

6.2 Control.Actions

6.2.1 Action

Listing 3: Action.java

```
package Control.Actions;

/*
 * Actions are constructed and executed by the inputController to
 * process moves and attacks.
 */
public abstract class Action
{

    /**
     * some things only happen after successful execution.
     */
    private boolean executed = false;
```

```
    /**
     * this method must be overeridden by all subclasses and implement
     * functionality to execute the action. the implementation must end by
     * setting isExecuted true if execution was successful.
     */
    public void execute ()
    {
    }

    /**
     * If the Action could be executed or not
     *
     * @return - true when the action could be executed
     */
    public boolean isExecuted ()
    {
        return this.executed;
    }

    /**
     * Setter for the isExecuted attribute
     *
     * @param executed - true when the action should be executed,
     *                   false when the action should not be revertable
     */
    public void setExecuted (boolean executed)
    {
        this.executed = executed;
    }

}
```

6.2.2 AttackAction

Listing 4: AttackAction.java

```
package Control.Actions;

import Control.Controllers.MapController;
import Control.Controllers.SceneController;
import Model.GameObject.Attribute;
import Model.GameObject.CreatureAdapter;
import Model.Map.AxialTransform;
import Model.Map.Tile;
import static Resource.Constants.GeneralConstants.ActionConstants.DIVIDE_WITH_HUNDRED;
import static Resource.Constants.GeneralConstants.ActionConstants.SECOND_LAST_TILE;
import Resource.Constants.GeneralConstants.GameObjectConstants;
import Resource.Constants.GeneralConstants.MapConstants;
import java.util.ArrayList;

/*
 * The AttackAction implements damage calculation and attributes
 * which are the same for melee and ranged attacks.
 * Differences between the subclasses are implemented there.
 */
public abstract class AttackAction extends Action
{

    /**
     * If the attacker has the Knockback attribute its effect is applied as a
     * moveAction to be reveretable. If the attacker doesn't have the attribute
     * this is not used.
     */
    private MoveAction _knockbackMove = null;

    /**
```

```
31      * _attacker is set in constructor and is the Creature that initiated the
32      * attack
33      */
34     private CreatureAdapter _attacker = null;

36     /**
37      * _defender is set in constructor and is the Creature that is targeted by
38      * the attack
39      */
40     private CreatureAdapter _defender = null;

42     /**
43      * tile the defender is on, when the attack is started. saved for reverting
44      * defender death (which removes the defender from the map)
45      */
46     private Tile _defenderOnAttackStartTile = null;

48     /**
49      * tile the attacker is on, when the attack is started. saved for reverting
50      * attacker death on melee attacks (which removes the defender from the map)
51      */
52     private Tile _attackerOnAttackStartTile = null;

54     /**
55      * the damage dealt to the defender. saved for reverting
56      */
57     private int _damageDealt = 0;

59     /**
60      * if the attacker has the Leech attribute they are healed. the amount
61      * healed is stored for reverting
62      */
63     private int _attackerHealed = 0;

65     /**
66      * Constructs a new AttackAction
67      *
68      * @param attacker Attacking Creature
69      * @param defender Defending Creature
70      */
71     public AttackAction (CreatureAdapter attacker, CreatureAdapter defender)
72     {
73         this._attacker = attacker;
74         this._defender = defender;
75     }

77     /**
78      * getter for the Attacker Creature
79      *
80      * @return - the attacking creature
81      */
82     public CreatureAdapter getAttacker ()
83     {
84         return this._attacker;
85     }

87     /**
88      * Setter for the Attacker Creature
89      *
90      * @param attacker - the attacking creture
91      */
92     public void setAttacker (CreatureAdapter attacker)
93     {
94         this._attacker = attacker;
95     }
```

```
97      /**
98       * Getter for the Defender Creature
99       *
100      * @return - the defending creature
101      */
102     public CreatureAdapter getDefender ()
103     {
104         return this._defender;
105     }

107     /**
108      * Setter for the Defender Creature
109      *
110      * @param defender - the defending creature
111      */
112     public void setDefender (CreatureAdapter defender)
113     {
114         this._defender = defender;
115     }

117     /**
118      * Getter for the dealt damage
119      *
120      * @return - the dealt damage
121      */
122     public int getDamageDealt ()
123     {
124         return this._damageDealt;
125     }

127     /**
128      * Setter for the dealt damage
129      *
130      * @param damageDealt - the dealt damage
131      */
132     public void setDamageDealt (int damageDealt)
133     {
134         this._damageDealt = damageDealt;
135     }

137     /**
138      * Getter for the value the Attacker is healed
139      *
140      * @return - the value the _attacker is healed
141      */
142     public int getAttackerHealed ()
143     {
144         return this._attackerHealed;
145     }

147     /**
148      * Setter for the value the _attacker is healed
149      *
150      * @param attackerHealed - the value the _attacker should be healed
151      */
152     public void setAttackerHealed (int attackerHealed)
153     {
154         this._attackerHealed = attackerHealed;
155     }

157     /**
158      * Getter for the tile where the _attacker is on start
159      *
160      * @return - the tile where the _attacker is on start
161      */
162     public Tile getAttackerOnAttackStartTile ()
```

```
    {
        return this._attackerOnAttackStartTile;
    }

    /**
     * Setter for the tile where the _attacker is on start
     *
     * @param attackerOnAttackStartTile - the tile where the _attacker should be
     *                                    on start
     */
    public void setAttackerOnAttackStartTile (Tile attackerOnAttackStartTile)
    {
        this._attackerOnAttackStartTile = attackerOnAttackStartTile;
    }

    /**
     * Calculates damage based on (Ranged-)Attack value of the attacker and
     * PhyssRes value of thee defender, possibly modified by Attributes. Then
     * applies damage to defender and heals attacker (if applicable). then
     * applies Knockback if needed. If the defender dies, their health is not
     * reduced but they are removed from the map.
     */
    @Override
    public void execute ()
    {
        /*
         * only do anything if the _attacker hasn't yet attacked, isn't
         * paralyzed and has line of sight on the target.
         */
        if (!(_attacker.hasAttacked()) &&
                !(_attacker.hasAttribute(Attribute.Paralyzed)) &&
                (MapController.getInstance().getMap().lineOfSight
                (_attacker, _defender)))
          {
            int damageToDeal = 0;
            /**
             * Determining damage to be dealt, depending on melee/ranged attack
             * damage is procentually reduced by PhysRes. This is not a bunch of
             * magic numbers but a damage formula.
             */
            if (MeleeAttackAction.class.isInstance(this))
              {
                damageToDeal =
                        Math.round((_attacker.getAttack()
                        * (1 - (_defender._getPhysRes() / DIVIDE_WITH_HUNDRED))));
              }
            else if (RangedAttackAction.class.isInstance(this))
              {
                damageToDeal =
                        Math.round((_attacker.getRangedAttack()
                        * (1 - (_defender._getPhysRes() / DIVIDE_WITH_HUNDRED))));
              }

            /**
             * Checking for attribute interactions
             */
            if (_defender.hasAttribute(Attribute.Undead) &&
                    !_attacker.hasAttribute(Attribute.Undead) &&
                    !_attacker.hasAttribute(Attribute.UndeadSlayer))
              {
                damageToDeal = 0;
              }
            else
              {
                if (_attacker.hasAttribute(Attribute.Leech))
                  {
```

```
                    damageToDeal += GameObjectConstants.LEECH_BONUS_DAMAGE;
                    this._attackerHealed =
                            _attacker.restoreHealth
                            (GameObjectConstants.LEECH_ATTACKER_HEAL);
                }
            }

        /*
         * applying damage
         */
        this._damageDealt = _defender.reduceHealth(damageToDeal);

        if (this._damageDealt == -1)
          //defender lives
          {
            _damageDealt = damageToDeal;
            /*
             * knockback implementationhere because it should be done after
             * applying damage (dead creatures dont need to be knocked back)
             */
            if (_attacker.hasAttribute(Attribute.Knockback))
              {

                ArrayList<Tile> attackLine = MapController.getInstance()
                        .getMap().getLine(
                                (_attacker).getTile(), _defender.getTile()
                        );

                /*
                 * second last tile in the list is the tile that an attack
                 * came from because the last tile is the tile the _defender
                 * currently is on. there should not be indexOutOfBounds
                 * exceptions since the _attacker and _defender will always
                 * be
                 * on different tiles and if they are adjacent the tile will
                 * be the attackerTile
                 */
                Tile attackedFrom = attackLine.get(attackLine.size() - SECOND_LAST_TILE );
                AxialTransform knockbackDirection = null;

                /*
                 * determine direction the attack came from
                 */
                for (int i = 0; i < MapConstants.AXIAL_DIRECTIONS.length; i++)
                  {
                    if (AxialTransform.add
                        (attackedFrom.getAxialTransform(),
                        MapConstants.AXIAL_DIRECTIONS[i])
                        .hashCode() == _defender.getTile().hashCode())
                      {
                        knockbackDirection = MapConstants.AXIAL_DIRECTIONS[i];
                        break;
                      }
                  }

                _defender.setMovementpoints(_defender.getMovementpoints() + 1);

                if (MapController.getInstance().getMap().getTile(
                        AxialTransform.add(
                                _defender.getTile().getAxialTransform(),
                                knockbackDirection)
                    ).getGameObject() == null)
                  {
                    this._knockbackMove =
                        new MoveAction(MapController.getInstance().getMap()
                            .getTile(
```

```
                                    AxialTransform.add(
                                        _defender.getTile().getAxialTransform(),
                                            knockbackDirection))
                                    , _defender);
                        _knockbackMove.execute();
                        if (!_knockbackMove.isExecuted())
                          {
                            _defender.setMovementpoints
                                (_defender.getMovementpoints() - 1);
                          }
                      }
                  }
              }
            else
              {
                this._defenderOnAttackStartTile = _defender.getTile();
              }
            _attacker.setHasAttacked(true);
            SceneController.getInstance().refresh();
            this.setExecuted(true);
          }
    }

}
```

6.2.3 AttackMoveAction

Listing 5: AttackMoveAction.java

```
package Control.Actions;

import Model.GameObject.CreatureAdapter;
import Model.Map.Tile;

/*
 * Combines a moveAction and an attackAction into one action, so an enemy
 * creature in moveRange can be approached and attacked as one action.
 */
public class AttackMoveAction extends Action
{

    /**
     * The Attack that will be taken in this AttackMoveAction
     */
    private AttackAction _thisAttack = null;

    /**
     * The Move that will be taken in this AttackMoveAction
     */
    private MoveAction _thisMove = null;

    /**
     *Constructs a new AttackMoveAction
     *
     * @param attacker - the attacking creature
     * @param moveTarget - a tile as a target
     * @param thisAttack - the attack that should be taken
     */
    public AttackMoveAction (
            CreatureAdapter attacker, Tile moveTarget, AttackAction thisAttack)
    {
        this._thisMove = new MoveAction(moveTarget, attacker);
        this._thisAttack = thisAttack;
    }
```

```
    /**
     * Execute the Move and the Attack Action of this Action
     */
    @Override
    public void execute ()
    {
        this._thisMove.execute();
        this._thisAttack.execute();
        setExecuted(true);
    }

}
```

6.2.4 ManaPickupAction

Listing 6: ManaPickupAction.java

```
package Control.Actions;

import Control.Controllers.MapController;
import Model.GameObject.ManaCrystal;
import Model.Player;

/**
 * A ManaPickupAction is called when a player picks up a manacrystal.
 */
public class ManaPickupAction extends Action
{

    /**
     * Crystal that is picked up
     */
    private ManaCrystal _pickedUpCrystal = null;

    /**
     * Player, that picked up the Crystal
     */
    private Player _pickingUpPlayer = null;

    /**
     * A ManaPickupAction is called when a player picks up a manacrystal.
     *
     * @param p The player currently picking up a manacrystal
     * @param c The picked up manacrystal
     */
    public ManaPickupAction (Player p, ManaCrystal c)
    {
        this._pickedUpCrystal = c;
        this._pickingUpPlayer = p;
    }

    /**
     * adds mana to player, then removes crystal
     */
    @Override
    public void execute ()
    {
        this._pickingUpPlayer.restoreMana(this._pickedUpCrystal.getManaGain());
        if (this._pickedUpCrystal.getTile() != null)
          {
            MapController.getInstance()
                    .removeObjectFromMap(this._pickedUpCrystal.getTile());
          }
        this.setExecuted(true);
    }
```

```
}
```

6.2.5 ManaPickupMoveAction

Listing 7: ManaPickupMoveAction.java

```
package Control.Actions;

import Control.Controllers.GameController;
import Model.GameObject.CreatureAdapter;
import Model.GameObject.ManaCrystal;
import Model.Map.Tile;

/**
 * A ManaPickupAction is called when a player picks up a manacrystal by
 * moving onto it.
 */
public class ManaPickupMoveAction extends Action
{

    /**
     * the move, executed to reach the tile that has a manaCrystal
     */
    private MoveAction _thisMove = null;

    /**
     * the PickupAction that restores mana to the player.
     */
    private ManaPickupAction _thisPickup = null;

    /**
     * A ManaPickupAction is called when a player picks up a manacrystal by
     * moving onto it.
     *
     * @param tam The tile the player is moving onto.
     * @param c   The creature he is using to move there.
     */
    public ManaPickupMoveAction (Tile tam, CreatureAdapter c)
    {
        _thisMove = new MoveAction(tam, c);
        _thisPickup
                = new ManaPickupAction(
                        GameController.getInstance().getCurrentPlayerController()
                                .getPlayer(c.getOwner()),
                        (ManaCrystal) tam.getGameObject()
                );
    }

    /**
     * executes pickup action, then move-action
     */
    @Override
    public void execute ()
    {
        if (_thisMove.isValid())
          {
            _thisPickup.execute();
          }
        _thisMove.execute();
        this.setExecuted(true);
    }

}
```

6.2.6 MeleeAttackAction

Listing 8: MeleeAttackAction.java

```java
package Control.Actions;

import Model.GameObject.CreatureAdapter;
import static Resource.Constants.GeneralConstants.ActionConstants.ATTACKER_LIVES;
import static Resource.Constants.GeneralConstants.ActionConstants.DIVIDE_WITH_HUNDRED;

/**
 * This Action applies MeleeAttacks.
 * A MeleeAttackAction is calles when a creature attacks a creature that is
 * next to it.
 */
public class MeleeAttackAction extends AttackAction
{

    /**
     * The damage that is taken by the attacker.
     */
    private int _damageTakenByAttacker = 0;

    /**
     * A MeleeAttackAction is calles when a creature attacks a creature that is
     * next to it.
     *
     * @param attacker the attacking creature.
     * @param defender the defending creature.
     */
    public MeleeAttackAction (CreatureAdapter attacker, CreatureAdapter defender)
    {
        super(attacker, defender);
    }

    /**
     * executes super.execute for damage calculation towards the defender. Also
     * deals damage to the attacker and removes them from the map if killed.
     */
    @Override
    public void execute ()
    {
        int damageToAttacker
                = Math.round((getDefender().getAttack() *
                        (1 - (getAttacker()._getPhysRes() / DIVIDE_WITH_HUNDRED))));

        super.execute();

        if (this.isExecuted())
          //Only if the attack was successfully executed, the attacker takes damage
          {

            this._damageTakenByAttacker
                    = getAttacker().reduceHealth(damageToAttacker);

            if (_damageTakenByAttacker == ATTACKER_LIVES) //attacker lives
              {
                _damageTakenByAttacker = damageToAttacker;
              }
            else //attacker dies
              {
                this.setAttackerOnAttackStartTile(getAttacker().getTile());
              }
          }
    }

}
```

6.2.7 MoveAction

Listing 9: MoveAction.java

```java
package Control.Actions;

import Control.Controllers.GameController;
import Control.Controllers.MapController;
import Control.Controllers.SceneController;
import Model.GameObject.Attribute;
import Model.GameObject.CreatureAdapter;
import Model.Map.Pathfinding;
import Model.Map.Tile;
import java.util.ArrayList;

/**
 * When this action is called, all necessities to move have already been checked
 * (specifically if there is a viable path available), except if the Creature
 * has sufficient MovementPoints, which is done here.
 */
public class MoveAction extends Action
{

    /**
     * The Creature that should be moved by this action
     */
    private CreatureAdapter _movedCreature = null;

    /**
     * the tile where the creature is after the moveAction
     */
    private Tile _tileAfterMove = null;

    /**
     * the used MovementPoints while this MoveAction
     */
    private int _usedMP = 0;

    /**
     * Constructs a new MoveAction
     *
     * @param tam - the Tile where the creature should be after the move
     * @param c   - the creature that should be moved by this MoveAction
     */
    public MoveAction (Tile tam, CreatureAdapter c)
    {
        this._movedCreature = c;
        this._tileAfterMove = tam;
    }

    /**
     * Moves the creature from its Tile to the given Tile, when it has not
     * attacked in this turn yet and when this creature is not paralyzed. When
     * the Movement Points of the creature are not enough to take this action,
     * the creature will not be moved. Flying Creatures could be attacked by
     * enemy creature with the attribute AirInterception.
     */
    @Override
    public void execute ()
    {
        if (this.isValid())
          {
            if (_movedCreature.hasAttribute(Attribute.Flying))
              {
                //if the moving creature is flying, it might be attacked by a
                //creature with air interception, during the move.
```

```
                for (CreatureAdapter creature : GameController.getInstance()
                        .getCurrentTurnController().getEnemyPlayer()
                                                        .getMyCreatures())
                  {
                    if (creature.hasAttribute(Attribute.AirInterception))
                      {
                        //air interception attack implementation
                        boolean airInterceptAttack = false;
                        ArrayList<Tile> attackArea =
                                MapController.getInstance().getMap()
                                    .getTilesInRange
                                    (creature.getTile(), creature.getAttackRange());

                        for (Tile pathTile : Pathfinding.getInstance().getPath())
                          {
                            if (attackArea.contains(pathTile))
                              {
                                airInterceptAttack = true;
                              }
                          }

                        if (airInterceptAttack)
                          {
                            RangedAttackAction airInterceptionAttack =
                                    new RangedAttackAction
                                    (creature, this._movedCreature);
                            airInterceptionAttack.execute();
                            creature.setAirInterceptionAttack(true);
                          }
                      }
                    creature.setAirInterceptionAttack(false);
                  }
              }
            if (_movedCreature.getTile() != null)
              {
                this._movedCreature.setMovementpoints
                    (this._movedCreature.getMovementpoints() - this._usedMP);
                MapController.getInstance().moveObject(_movedCreature,
                                                        _tileAfterMove);
              }
            this._movedCreature.getTooltip().hide(this._movedCreature.getTooltip());
            SceneController.getInstance().refresh();
            this.setExecuted(true);
          }
      }

      /**
       * checks if the move is valid, meaning if the creature has enough movement
       * points to reach its destination.
       *
       * @return true if valid
       */
      public boolean isValid ()
      {
          if ((!(_movedCreature.hasAttacked())) &&
                  !(this._movedCreature.hasAttribute(Attribute.Paralyzed)))
            {
              this._usedMP = Pathfinding.getInstance().calcMovementPoints(
                      this._movedCreature.getTile(),
                      this._tileAfterMove, false, this._movedCreature);

              if (this._usedMP <= this._movedCreature.getMovementpoints())
                {
                  return true;
                }
            }
```

```
        return false;
    }

}
```

6.2.8 RangedAttackAction

Listing 10: RangedAttackAction.java

```
package Control.Actions;

import Model.GameObject.Attribute;
import static Model.GameObject.Attribute.Paralysing;
import static Model.GameObject.Attribute.Paralyzed;
import Model.GameObject.CreatureAdapter;

/**
 * This Action applies Ranged Attacks.
 * A RangedAttackAction is called when a creature attacks a creature that is not
 * directly next to it and it has enought range to still attack it.
 */
public class RangedAttackAction extends AttackAction
{

    /**
     * true, if this attack applied Paralyzing to the defender on execution.
     * saved for reverting.
     */
    private boolean _paralyzingApplied = false;

    /**
     * A RangedAttackAction is called when a creature attacks a creature that is
     * not directly next to it and it has enought range to still attack it.
     *
     * @param attacker the attacking creature.
     * @param defender the defending creature.
     */
    public RangedAttackAction (CreatureAdapter attacker, CreatureAdapter defender)
    {
        super(attacker, defender);
    }

    /**
     * executes super, then applies Ranged-Attack-only attributes.
     */
    @Override
    public void execute ()
    {
        super.execute();
        if (this.isExecuted())
          {
            getAttacker().onRangedAttack();
            if (getAttacker().hasAttribute(Paralysing) &&
                    !(getDefender().hasAttribute(Paralyzed)) &&
                    !(getDefender().hasAttribute(Attribute.UndeadSlayer)))
              {
                this._paralyzingApplied = true;
                getDefender().addAttribute(Paralyzed);
              }
          }
    }

}
```

6.3 Control.Controllers

6.3.1 AnimationController

Listing 11: AnimationController.java

```java
package Control.Controllers;

import Model.Map.Tile;
import View.UIElements.MapView;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Platform;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

/**
 * The AnimationController is the superhub for all animation related mechanics.
 */
public class AnimationController
{
    /**
     * The instance of the AnimationCOntroller to make it a Singleton.
     */
    private static AnimationController _instance = new AnimationController();

    /**
     * The current MapView of the Game.
     */
    private MapView _mapView = null;

    /**
     * Getter fot the Instance of this Singleton.
     *
     * @return - instance
     */
    public static AnimationController getInstance ()
    {
        return _instance;
    }

    /**
     * Setter for the current MapView.
     *
     * @param mapView - the current view on the map.
     */
    public void setMapView (MapView mapView)
    {
        _mapView = mapView;
    }

    /**
     * Plays the animation found on the given Path on a given Tile.
     *
     * @param path - path to the animation data
     * @param tile - the tile where the animation should be played.
     */
    public void playAnimationOnTile (String path, Tile tile)
    {
        int x = (int) (_mapView.getTileView(tile).getX());
        int y = (int) (_mapView.getTileView(tile).getY());
        createAnimation(x, y, path);
    }
```

```
    /**
     * Creates a animation with the source of the given path.
     *
     * @param x - width of the animation
     * @param y - height of the animation
     * @param path - path to animation source
     */
    private void createAnimation (int x, int y, String path)
    {
        try
          {
            ImageView animation = new ImageView(new Image(
                                                    new FileInputStream(path)));

            animation.setX(x);
            animation.setY(y);
            Platform.runLater(() ->
              {
                _mapView.getChildren().add(animation);
              });
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(AnimationController.class.getName())
                                                   .log(Level.SEVERE, null, ex);
          }
    }
}
```

6.3.2 GameController

Listing 12: GameController.java

```
package Control.Controllers;

import Control.Game;
import Control.Network.GameClient;
import Control.Network.GameServer;
import Model.Game.FluxMeter;
import static Resource.Constants.GeneralConstants.ControllerConstants.NOT_MULTI;
import static Resource.Constants.GeneralConstants.ControllerConstants.NO_CLIENT;

/**
 * The GameController is the superhub for all Game related classes.
 */
public class GameController
{

    /**
     * Instance of the GameController to make it a Singleton.
     */
    private static GameController _instance = new GameController();

    /**
     * The GameClient to provide networking for multiplay.
     */
    private GameClient _client = null;

    /**
     * The GameServer to provide networking for multiplay.
     */
    private GameServer _server = null;

    /**
     * The current Game.
```

```
33      */
34     private Game _game = null;

36     /**
37      * Singleton stuff.
38      *
39      * @return this Game Controller
40      */
41     public static GameController getInstance ()
42     {
43         return _instance;
44     }

46     /**
47      * Getter for the GameCLient.
48      *
49      * @return - GameClient
50      */
51     public GameClient getClient ()
52     {
53         if (this._game.isMultiplayer())
54           {
55             if (this._client != null)
56               {
57                 return this._client;
58               }
59             System.err.println(NO_CLIENT);
60           }
61         System.err.println(NOT_MULTI);

63         return null;
64     }

66     /**
67      * Used to access the currently running Game.
68      *
69      * @return the currently active Game.
70      */
71     public Game getCurrentGame ()
72     {
73         if (this._game != null)
74           {
75             return this._game;
76           }
77         return null;
78     }

80     /**
81      * Used to access the Turn Controller of the running game.
82      *
83      * @return the currently active Turn Controller.
84      */
85     public TurnController getCurrentTurnController ()
86     {
87         return this._game.getTurnController();
88     }

90     /**
91      * Used to access the Player Controller of the running game.
92      *
93      * @return the currently active Player Controller.
94      */
95     public PlayerController getCurrentPlayerController ()
96     {
97         return this._game.getPlayerController();
98     }
```

```
    public void setClient (GameClient _client)
    {
        this._client = _client;
    }

    /**
     * Creates a new game, and starts the thread.
     *
     * @param multiplayer True if this game is being played over any network.
     */
    public void initGame (boolean multiplayer)
    {
        this._game = new Game(multiplayer);
    }

    public void startGame ()
    {
        this._game.start();

        MapController.getInstance().init();
        FluxMeter.getInstance().init();
    }

    public void shutDown ()
    {
        if (this.getClient() != null)
          {
            this.getClient().shutDown();
          }
        if (this._server != null)
          {
            this._server.shutDown();
          }
        if (this.getCurrentGame() != null)
          {
            this.getCurrentGame().interrupt();
          }

        SoundController.getInstance().reset();
    }

}
```

6.3.3 InputController

Listing 13: InputController.java

```
package Control.Controllers;

import Control.Actions.*;
import Control.Network.Packets.Packet;
import Control.SemiControllers.Highlighter;
import Control.SemiControllers.Selector;
import Exceptions.UnexpectedCardException;
import Exceptions.UnexpectedGameObjectException;
import Model.Card.AllCards.Spells.AttackSpell;
import Model.Card.AllCards.Spells.Spell;
import Model.Card.AllCards.Summons.Summon;
import Model.Card.ICard;
import Model.GameObject.*;
import Model.Map.Pathfinding;
import Model.Map.Tile;
import static Resource.Constants.GeneralConstants.ControllerConstants.MOVE;
import static Resource.Constants.GeneralConstants.ControllerConstants.SPACE;
import java.util.HashMap;
```

```
/**
 * The InputController procsses Inputs based on selected tiles It is called by
 * the selector, whenever a primary or secondary tile is selected or deselected
 * or when a Card is selected.
 */
public class InputController
{

    /**
     * Instance of the InputController to make it a Singleton.
     */
    private static InputController _instance = new InputController();

    /**
     * Getter for the instance of the INputController.
     *
     * @return - InputController instance
     */
    public static InputController getInstance ()
    {
        return _instance;
    }

    /**
     * Determines if the given tile holds a CreatureAdapter or a Obstacle or
     * nothing.
     *
     * @param primarySelectedTile The currently selected tile.
     *
     * @return - true if the processed Tile holds a CreatureAdapter.
     *
     * @throws UnexpectedGameObjectException if the Object on the tile is from a
     *                                       class it is not supposed to be
     *                                       from.
     */
    public boolean processPrimaryTileSelection (Tile primarySelectedTile) throws
            UnexpectedGameObjectException
    {
        switch (this.whatObjectAmI(primarySelectedTile.getGameObject()))
          {
            case noGO:
                break;

            case Obstacle:
                break;

            case ManaCrystal:
                break;

            case CreatureAdapter:
              {
                return true;
              }
            case nonDefinedObject://Unexpected GameObject on primary Tile
                throw new UnexpectedGameObjectException();
            default:
                break;
          }
        return false;
    }

    /**
     * This method is to be used, when the primaryTile is known to have a
     * Creature on it and the secondaryTile is not null but it is unknown if the
     * active player is the owner of the Creature on the primary tile.
```

```
85      *
86      * @param primaryTile        The first tile selected.
87      * @param secondaryTile      The second tile selected.
88      * @param activePlayerNumber the PlayerNumber of the Player that is
89      *                           currently taking action. In local games this is
90      *                           the currentPlayerNumber.
91      *
92      * @throws Exceptions.UnexpectedGameObjectException if the Object on any of
93      *                                                  the tiles is from a
94      *                                                  class it should not be
95      *                                                  from.
96      */
97     public void processTileSelection (Tile primaryTile, Tile secondaryTile,
98             int activePlayerNumber) throws UnexpectedGameObjectException
99     {
100        if (!(((CreatureAdapter) primaryTile.getGameObject())
101                .isOwnedBy(activePlayerNumber)))
102          {
103            //Enemy creature on primary Tile
104          }
105        else
106          {
107            this.processTileSelection(primaryTile, secondaryTile, true);
108          }
109    }

111    /**
112     * This method is to be used, if the primary tile is known to have a
113     * creature, which belongs to the active player (the player initiating the
114     * action) and the secondary Tile is not null.
115     *
116     * @param primaryTile   the tile selected first.
117     * @param secondaryTile the tile selected second.
118     * @param send          True if this selection has been provided by a
119     *                      packet.
120     *
121     * @throws UnexpectedGameObjectException if the Object on any of
122     *                                       the tiles is from a
123     *                                       class it should not be
124     *                                       from.
125     */
126    public void processTileSelection (Tile primaryTile, Tile secondaryTile,
127            boolean send) throws UnexpectedGameObjectException
128    {
129        //Checking GameObject on Secondary Tile
130        switch (this.whatObjectAmI(secondaryTile.getGameObject()))
131          {
132            case noGO: //Move Action
133                (new MoveAction(secondaryTile, ((CreatureAdapter) primaryTile
134                                .getGameObject()))).execute();
135                break;

137            case Obstacle: //No action
138                break;

140            case ManaCrystal: //pickup Action && Move Action
141                (new ManaPickupMoveAction(secondaryTile,
142                                          ((CreatureAdapter) primaryTile.
143                                                  getGameObject()))).execute();
144                break;

146            case CreatureAdapter://Creature
147                if (!(((CreatureAdapter) secondaryTile.getGameObject())
148                        .isOwnedBy(((CreatureAdapter) primaryTile.getGameObject())
149                                .getOwner())))
150                  {
```

```
                    /*
                     * enemy
                     * Attack Action || MoveToAttack Action ||
                     * MoveAsCloseAsPossible Action as deetermined by
                     * determineAttaccckAction()
                     */
                    determineAttackAction(primaryTile, secondaryTile).execute();
                  }
                else
                  { //frienly creatures on both primary and secondary tile
                    //
                      {
                        //firendly non-PC creature on primary and secondary tiles
                      }
                  }
                break;
            case nonDefinedObject: //Unexpected GameObject on secondary Tile
                throw new UnexpectedGameObjectException();
            default:
                break;
          }

        if (GameController.getInstance().getCurrentGame().isMultiplayer() && send)
          {
            String packetString = MOVE + primaryTile.getAxialTransform()
                    .getH() + SPACE + primaryTile.getAxialTransform()
                            .getD() + SPACE + secondaryTile.getAxialTransform()
                            .getH() + SPACE + secondaryTile.getAxialTransform().getD();
            GameController.getInstance().getClient()
                    .sendPackage(new Packet(packetString));
          }
    }

    /**
     * Highlights all valid target tiles of a spell
     *
     * @throws UnexpectedCardException if the currently selected card is not of
     *                                 any class it should be of.
     */
    public void processCardSelection () throws UnexpectedCardException
    {
        ICard c = Selector.getInstance().getSelectedCard();
        if (c != null)
          {
            Highlighter.getInstance().unhighlight();
            if (Spell.class
                    .isInstance(c))
              {
                //Spell
                if (AttackSpell.class
                        .isInstance(c))
                  {
                    /*
                     * Iterates over enemy creatures. Highlights their tile if
                     * they are in range of the spell and - if the spell
                     * requires LineOfSight - visible to the friendly Wizard.
                     */
                    for (GameObject go
                         : GameController.getInstance()
                                    .getCurrentTurnController().getEnemyPlayer()
                                    .getMyCreatures())
                      {
                        if (MapController.getInstance().getMap()
                                .getTilesInRangeExceptObstacles(GameController
                                        .getInstance().getCurrentTurnController()
                                        .getCurrentPlayer().getCharacterInstance()
```

```
                                        .getTile(), ((AttackSpell) c).getCastRange())
                                .contains(go.getTile()))
                            {
                              if (((Spell) c).requiresLineOfSight())
                                {
                                  if (MapController.getInstance().getMap()
                                          .lineOfSight(GameController.getInstance()
                                                  .getCurrentTurnController()
                                                  .getCurrentPlayer()
                                                  .getCharacterInstance(), go.getTile()))
                                    {
                                      Highlighter.getInstance()
                                              .highlightSpellTarget(go.getTile());
                                    }
                                }

                              else
                                {
                                  Highlighter.getInstance()
                                          .highlightSpellTarget(go.getTile());
                                }

                            }
                        }

                      return;
                    }
                  Highlighter.getInstance()
                          .highlightSpellTarget(MapController.getInstance()
                                  .getMap().getTilesInRange(GameController.getInstance()
                                          .getCurrentTurnController().getCurrentPlayer()
                                          .getCharacterInstance().getTile(),
                                                          ((Spell) c).getCastRange()));
                }
              //no spell, summon?
              else if (Summon.class
                      .isInstance(c))
                {
                  for (Tile t
                       : MapController.getInstance()
                                  .getMap().getRing(GameController.getInstance()
                                          .getCurrentTurnController().getCurrentPlayer()
                                          .getCharacterInstance().getTile(), 1))
                    {
                      if (t.getGameObject() == null || ManaCrystal.class
                              .isInstance(t.getGameObject()))
                        {
                          Highlighter.getInstance().highlightSpellTarget(t);
                        }
                    }

                }
              else
                {
                  //neither spell nor summon
                  throw new UnexpectedCardException();
                }
            }
    }

    /**
     * Used for determining the type of a GameObject for usage in switch-case
     *
     * @param go GameObject to deetermine type of
     *
     * @return corresponding element of the GameObjects-enum
```

```
     */
    private GameObjects whatObjectAmI (GameObject go)
    {
        if (go == null)
          {
            return GameObjects.noGO;
          }
        else if (Obstacle.class
                .isInstance(go))
          {
            return GameObjects.Obstacle;

          }
        else if (ManaCrystal.class
                .isInstance(go))
          {
            return GameObjects.ManaCrystal;

          }
        else if (CreatureAdapter.class
                .isInstance(go))
          {
            return GameObjects.CreatureAdapter;

          }
        else
          {
            return GameObjects.nonDefinedObject;
          }
    }

    /**
     * this method determines which type of attack action to execute when
     * processTileSelection() already made sure there's a turn-player friendly
     * Creature on primary Tile and an enemy creature on secondary tile.
     *
     * @param primaryTile   The first tile selected.
     * @param secondaryTile The second tile selected.
     *
     * @return - determined action.
     */
    private Action determineAttackAction (Tile primaryTile, Tile secondaryTile)
    {
        Tile a = primaryTile;
        Tile b = secondaryTile;

        //if adjacent -> melee attack
        if (MapController.getInstance().getMap().getNeighbours(b).contains(a))
          {
            return new MeleeAttackAction(((CreatureAdapter) a.getGameObject()),
                                         ((CreatureAdapter) b.getGameObject()));
          }
        else
          {        //within attackRange of attacker and LineOfSight true
            if ((MapController.getInstance().getMap().getDistance(a, b) <=
                    ((CreatureAdapter) (a.getGameObject())).getAttackRange()) &&
                    (MapController.getInstance().getMap()
                            .lineOfSight(a.getGameObject(), b.getGameObject())))
              {
                return new RangedAttackAction(((CreatureAdapter) a.getGameObject()),
                        ((CreatureAdapter) b.getGameObject()));
              }
            else
              {
                /*
                 * determine a tile that is whithin the attacker's attack range
```

```
                    * from the defender and is closest to attackers current
                    * position
                    * Key: MP needed to reach tile; Value: Tile
                    */
                   HashMap<Integer, Tile> tilesInAttRange = new HashMap<>();

                   for (Tile t : MapController.getInstance().getMap()
                           .getTilesInRange(b, ((CreatureAdapter) (a.getGameObject()))
                                           .getAttackRange()))
                     {
                       if (MapController.getInstance().getMap()
                               .lineOfSight(a.getGameObject(), b.getGameObject()))
                         {
                           tilesInAttRange.put(Pathfinding.getInstance()
                                   .calcMovementPoints(a, t, false,
                                                   ((CreatureAdapter) (a.getGameObject()))), t);
                         }
                     }
                   //i starts at one because at this point theres no need to look
                   //for a 0-MP-move
                   for (int i = 1; i <= ((CreatureAdapter) (a.getGameObject()))
                        .getMovementpoints(); i++)
                     {
                       if (tilesInAttRange.containsKey(i))
                         {
                           if (((CreatureAdapter) (a.getGameObject()))
                                   .getAttackRange() == 1)
                             {//Attacker is melee fighter

                               return new AttackMoveAction(((CreatureAdapter) a
                                       .getGameObject()), tilesInAttRange.get(i),
                                                           new MeleeAttackAction(((
                               CreatureAdapter) a.getGameObject()),
                               ((CreatureAdapter) b.getGameObject()));
                             }
                           else
                             {//attacker has ranged  attack

                               return new AttackMoveAction(((CreatureAdapter) a
                                       .getGameObject()), tilesInAttRange.get(i),
                                                           new RangedAttackAction((
                               (CreatureAdapter) a.getGameObject()),
                               ((CreatureAdapter) b.getGameObject()));
                             }
                         }
                     }
                   /*
                    * if none off the above actions is returned,
                    * return a move action to the same tile the attacker is on,
                    * basically doing nothing.
                    * there probably is a better way off doing this but oh well...
                    * - Henning
                    */
                   return new MoveAction(a, ((CreatureAdapter) a.getGameObject()));
                 }
             }
     }

}
```

6.3.4 MapController

Listing 14: MapController.java

```
package Control.Controllers;
```

```
3  import Exceptions.MapInvalidException;
4  import Exceptions.MapTypeNotSupportedException;
5  import Model.GameObject.GameObject;
6  import Model.Map.*;
7  import static Resource.Constants.GeneralConstants.ControllerConstants.MAXIMUM_OF_TRIES;
8  import static Resource.Constants.GeneralConstants.ControllerConstants.NUMBER_OF_WIZARD_SPAWNS;
9  import Resource.Constants.GeneralConstants.MapConstants;

11 /**
12  * The MapController provides functionality to the map and Tiles, such as moving
13  * objects, it is the main way for Objects outside the map package to interact
14  * with the map and/or it's tiles.
15  */
16 public class MapController
17 {

19     /**
20      * instance for making the MapController a singleton.
21      */
22     private static MapController _instance = new MapController();

24     /**
25      * map this Controller controls.
26      */
27     private Map _map = null;

29     /**
30      * Getter for the instance of the singleton MapController.
31      *
32      * @return instance of the MapController.
33      */
34     public static MapController getInstance ()
35     {
36         return _instance;
37     }

39     /**
40      * Getter for the Map.
41      *
42      * @return - The current Map
43      */
44     public Map getMap ()
45     {
46         return this._map;
47     }

49     /**
50      * generates a map and sets wizardSpawns according to ControllerConstants.
51      */
52     public void init ()
53     {
54         int attempts = 0;
55         do
56           {
57             if (!(MapGenArgs.getFileName() == null) && !MapGenArgs.getFileName()
58                                                                    .equals(""))
59               {
60                 this._map = new Map(MapGenerator.loadMap(MapGenArgs.getFileName()));
61               }
62             else
63               {
64                 try
65                   {
66                     this._map = new Map(MapGenerator.generateMap(MapGenArgs
67                                                                  .getArgs()));
68                   }
```

```
                catch (MapTypeNotSupportedException e)
                  {
                    this._map = new Map(MapGenerator.loadMap(MapConstants
                                                              .DEFAULT_MAP));
                  }

                for (int i = 0; i < NUMBER_OF_WIZARD_SPAWNS; i++)
                  {
                    removeObjectFromMap(MapGenerator.getWizardSpawns()[i]);
                    /*
                     * if both wizards were placed on the map here,
                     *  this.wizardPath() would always return Integer.MAX_VALUE.
                     */
                  }
              }
          }
        while (!this.wizardPath() && attempts < MAXIMUM_OF_TRIES);

        if (attempts < MAXIMUM_OF_TRIES)
          {
            for (int i = 0; i < NUMBER_OF_WIZARD_SPAWNS; i++)
              {
                placeObjectOnMap(GameController.getInstance()
                    .getCurrentPlayerController().getPlayer(i)
                    .getCharacterInstance(), MapGenerator.getWizardSpawns()[i]);
                //Only after the map is checked for being valid, place wizards.
              }
          }
        else
          {
            throw new MapInvalidException();
          }
    }

    /**
     * Adds an Object to a Tile if it isn't already occupied, Logs to Console if
     * it is occupied and the Object isn't set.
     *
     * @param o the object that should be placed
     * @param t the tile where the object should be placed
     *
     * @return true if the given object could be placed on given tile, false if
     *         tile is null or there is a gameobject on given tile
     *
     */
    public boolean placeObjectOnMap (GameObject o, Tile t)
    {
        if ((t != null) && (t.getGameObject() == null))
          {
            t.setGameObject(o);
            o.setTile(t);
            return true;
          }
        else
          {
            StringBuilder sb = new StringBuilder();
            if (t == null)
              {
                /*
                 * sb.append("Attempted to place Object on Tile but failed.
                 * Tile with doesn't exist");
                 */
              }
            else
              {
//                sb.append("Attempted to place Object on Tile but failed
```

```
                    //. Tile with Transform ");
//                  sb.append(t.toString());
//                  sb.append(" already occupied");
                }
//              System.out.println(sb.toString());
              return false;
            }
    }

    /**
     * Removes the object on a tile if there is one.
     *
     * @param t the tile where a object should be removed
     *
     * @return true if the object on the given tile could be removed, false if
     *         there is no object or the tile is null.
     */
    public boolean removeObjectFromMap (Tile t)
    {

        if ((t != null) && (t.getGameObject() != null))
          {
            if (t.getGameObject() != null)
              {
                t.getGameObject().setTile(null);
              }
            t.setGameObject(null);

            return true;
          }
        return false;
    }

    /**
     * Moves an object from it's old position to a new position.
     *
     * @param o the object that should be moved
     * @param t the coordinates there the object should be moved to
     *
     * @return true if the object was moved, false if there is no object
     */
    public boolean moveObject (GameObject o, Tile t)
    {
        if (removeObjectFromMap(o.getTile()))
          {
            placeObjectOnMap(o, t);
            return true;
          }
        return false;
    }

    /**
     * Method to check, if on a randomly generated map, there is a path between
     * both spawns. This is checked to prevent situations where one wizard is
     * surrounded by obstacles.
     *
     * @return
     */
    private boolean wizardPath ()
    {
        if (Integer.MAX_VALUE > Pathfinding.getInstance()
            .calcMovementPoints(MapGenerator.getWizardSpawns()[0],
                MapGenerator.getWizardSpawns()[1], true, GameController
                    .getInstance().getCurrentPlayerController().getPlayer(0)
                                                        .getCharacterInstance()))
          {
```

```
            return true;
          }
        else
          {
            return false;
          }
    }

}
```

6.3.5 PlayerController

Listing 15: PlayerController.java

```
package Control.Controllers;

import Model.Player;
import Resource.Constants.GeneralConstants.ControllerConstants;

/**
 * The PlayerController is a Singleton, controlclass for the players.
 */
public class PlayerController
{

    /**
     * An Array that holds the Player of the game.
     */
    private Player[] _players = new Player[ControllerConstants.NUMBER_OF_PLAYERS];

    /**
     * The Instance for making the PlayerController a Singleton.
     */
    private static final PlayerController _instance = new PlayerController();

    /**
     * Getter fo the instance of the singleton PlayerController.
     *
     * @return - the PlayerController instance
     */
    public static PlayerController getInstance ()
    {
        return _instance;
    }

    /**
     * Getter for the Player with the given ownerNumber.
     *
     * @param ownerNumber - the Number of the Player
     *
     * @return the Player with the given ownerNumber
     */
    public Player getPlayer (int ownerNumber)
    {
        return _players[ownerNumber];
    }

    /**
     * Initialises the Player Array with two new Players.
     */
    public void init ()
    {
        for (int i = 0; i < _players.length; i++)
          {
            _players[i] = new Player(Integer.toString(i), i);
          }
```

```
    }
}
```

6.3.6 SceneController

Listing 16: SceneController.java

```
package Control.Controllers;

import Resource.Constants.GeneralConstants.ControllerConstants;
import Resource.Dimensions;
import View.Scenes.ChaosScene;
import View.Scenes.GameScene;
import View.Scenes.GameSetupSceneMP;
import View.Scenes.MenuScene;
import javafx.application.Platform;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.stage.Stage;

/**
 * The Scenecontroller is the superhub of all Scene related classes.
 */
public class SceneController
{

    /**
     * The instance for making the SceneController a Singleton.
     */
    private static final SceneController _instance = new SceneController();

    /**
     * The Current ChaosScene.
     */
    private ChaosScene _currentChaosScene = null;

    /**
     * The Current Scene.
     */
    private Scene _currentScene = null;

    /**
     * The primary Stage.
     */
    private Stage _primaryStage = null;

    /**
     * this is used when the escape-key is pressed to check if a pause menu is
     * already open.
     */
    private boolean _isPaused = false;

    /**
     * private Constructor for the Singleton SecenController.
     */
    private SceneController ()
    {
        this._primaryStage = new Stage();
        this._currentChaosScene = new MenuScene();
        this._currentScene = new Scene(_currentChaosScene.getGroup(),
                                       Dimensions.getWidth(), Dimensions.getHeight());
        _primaryStage.setTitle(ControllerConstants.TITLE);
        _primaryStage.setScene(_currentScene);
        /*
         * this is needed because by default pressing the close button
         * of a window doesn't close all threads
```

```
60          */
61         _primaryStage.setOnCloseRequest(event ->
62           {
63             System.exit(0);
64           });
65     }

67     /**
68      * Getter for the instance of the SceneController.
69      *
70      * @return SceneController instance
71      */
72     public static SceneController getInstance ()
73     {
74         return _instance;
75     }

77     /**
78      * Getter for the Curren ChaosScene..
79      *
80      * @return - current ChaosScene.
81      */
82     public ChaosScene getCurrentChaosScene ()
83     {
84         return _currentChaosScene;
85     }

87     /**
88      * Setter for the Scene.
89      *
90      * @param s The Scene that is to be displayed
91      */
92     public void setScene (ChaosScene s)
93     {
94         s.scaleAllSceneNodesToMonitorSize();
95         _currentChaosScene = s.refresh();
96         _currentScene.setRoot(_currentChaosScene.getGroup());
97     }

99     /**
100     * Sets the Scene of the SceneController to the GameScene.
101     */
102    public void setSceneToGameScene ()
103    {
104        this.setScene(new GameScene());
105    }

107    /**
108     * Sets Scene of thre SceneController to the Multiplayer GameSetupScene.
109     */
110    public void setSceneToGameSetupSceneMP ()
111    {
112        this.setScene(new GameSetupSceneMP(GameController.getInstance()
113                .getClient().getPlayerNumber()));
114    }

116    /**
117     * Getter for the primary Stage.
118     *
119     * @return - primary Stage.
120     */
121    public Stage getPrimaryStage ()
122    {
123        return _primaryStage;
124    }
```

```
    /**
     * Adds a given Node to the Scene.
     *
     * @param n - the node that has to be added
     */
    public void addNode (Node n)
    {
        if (!this._currentChaosScene.getGroup().getChildren().contains(n))
          {
            this._currentChaosScene.addNodes(n);
            refresh();
          }

    }

    /**
     * Removes a given node from the Scene.
     *
     * @param n - the node thas has to be removed
     */
    public void removeNode (Node n)
    {
        if (this._currentChaosScene.getGroup().getChildren().contains(n))
          {
            this._currentChaosScene.removeNode(n);
            refresh();
          }

    }

    /**
     * Getter for the current GameScene.
     *
     * @return - the current GameScene
     */
    public GameScene getCurrentGameScene ()
    {
        if (GameScene.class.isInstance(this._currentChaosScene))
          {
            return (GameScene) this._currentChaosScene;
          }
        return null;
    }

    /**
     * Getter for the current Scene.
     *
     * @return - the current Scene.
     */
    public Scene getCurrentScene ()
    {
        return this._currentScene;
    }

    /**
     * Getter that says if the Game is paused or not.
     *
     * @return - true if the game is paused.
     */
    public boolean isIsPaused ()
    {
        return _isPaused;
    }

    /**
     * Setter if the game is paused.
```

```
     *
     * @param isPaused True if the game has been paused.
     */
    public void setIsPaused (boolean isPaused)
    {
        this._isPaused = isPaused;
    }

    /**
     * Refreshs the current ChaosScene.
     *
     */
    public void refresh ()
    {
        Platform.runLater(() ->
          {
            setScene(_currentChaosScene.refresh());
          });
    }
}
```

6.3.7 SoundController

Listing 17: SoundController.java

```
package Control.Controllers;

import View.UIElements.MusicAndEffect;

/**
 * The SoundController is the superhub for all Sound related stuff.
 */
public class SoundController
{

    /**
     * The instance to make the SoundCOntroller a Singleton.
     */
    private static SoundController instance = new SoundController();

    /**
     * Music and Effects of the Game.
     */
    private MusicAndEffect _music = new MusicAndEffect();

    /**
     * Singleton getter.
     *
     * @return The instance of this Singleton.
     */
    public static SoundController getInstance ()
    {
        return instance;
    }

    /**
     * Getter.
     *
     * @return the musicplayer.
     */
    public MusicAndEffect getMusic ()
    {
        return this._music;
    }

    /**
```

```
     * Stops the current music and effects and starts again.
     */
    public void reset()
    {
        this._music.stopAll();
        this._music = new MusicAndEffect();
    }

}
```

6.3.8 TurnController

Listing 18: TurnController.java

```
package Control.Controllers;

import Control.Actions.Action;
import Control.SemiControllers.Selector;
import Model.GameObject.Attribute;
import Model.GameObject.CreatureAdapter;
import Model.Player;
import Resource.Constants.GeneralConstants.ControllerConstants;
import static Resource.Constants.GeneralConstants.PlayerConstants.PLAYER_0;
import static Resource.Constants.GeneralConstants.PlayerConstants.PLAYER_1;
import java.util.ArrayList;

/**
 * The turnController stores the Active player and alternates turns between
 * players.
 */
public class TurnController
{

    /**
     * The Instance for making the TurnController a Singleton.
     */
    private static TurnController _instance = new TurnController();

    /**
     * stores the last actions for reverting them.
     */
    private ArrayList<Action> _lastActionsTaken = null;

    /**
     * Stores the current Player of the turn.
     */
    private Player _currentPlayer = null;


    /**
     * Getter for the Current Player of the Turn.
     *
     * @return - the current Player
     */
    public Player getCurrentPlayer ()
    {
        return this._currentPlayer;
    }

    /**
     * Getter for the current enemy Player.
     *
     * @return - the current enemy Player
     */
    public Player getEnemyPlayer ()
    {
```

```
        if (_currentPlayer == GameController.getInstance()
                                .getCurrentPlayerController().getPlayer(PLAYER_0))
          {
            return GameController.getInstance().getCurrentPlayerController()
                                                                .getPlayer(PLAYER_1);
          }
        else if (_currentPlayer == GameController.getInstance()
                                .getCurrentPlayerController().getPlayer(PLAYER_1))
          {
            return GameController.getInstance().getCurrentPlayerController()
                                                                .getPlayer(PLAYER_0);
          }
        return null;
    }

    /**
     * Getter for the instance of the Singleton TurnController.
     *
     * @return - the instance of the TurnController
     */
    public static TurnController getInstance ()
    {
        return _instance;
    }

    /**
     * sets Player0 as the starting active player.
     */
    public void init ()
    {
        this._lastActionsTaken = new ArrayList<Action>();
        this._currentPlayer = GameController.getInstance()
                                .getCurrentPlayerController().getPlayer(PLAYER_0);
    }

    /**
     * executes onEndTurn, then switches the active player with the current
     * enemy player.
     */
    public void nextPlayer ()
    {
        onEndTurn();
        if (_currentPlayer == GameController.getInstance()
                                .getCurrentPlayerController().getPlayer(PLAYER_0))
          {
            this._currentPlayer = GameController.getInstance()
                                  .getCurrentPlayerController().getPlayer(PLAYER_1);
          }
        else if (_currentPlayer == GameController.getInstance()
                                  .getCurrentPlayerController().getPlayer(PLAYER_1))
          {
            this._currentPlayer = GameController.getInstance()
                                  .getCurrentPlayerController().getPlayer(PLAYER_0);
          }
    }

    /**
     * When a player ends their turn their creatures refill movementpoints, gain
     * attacks, remove paralyzation, the last Actions list is cleared so the
     * next player cant revert their enemy's actions the player draws a card and
     * gains Mana pere turn according to Constants.
     */
    private void onEndTurn ()
    {
        for (CreatureAdapter c : _currentPlayer.getMyCreatures())
          {
```

```
            c.setMovementpoints(c.getMaxMvp());
            c.setHasAttacked(false);
            if (c.hasAttribute(Attribute.Paralyzed))
              {
                c.removeAttribute(Attribute.Paralyzed);
              }
          }
        this._lastActionsTaken.clear();

        this._currentPlayer.getHand().drawCard();
        this._currentPlayer.restoreMana(ControllerConstants.DEFAULT_MANA_PER_TURN);
        Selector.getInstance().deselectCard();
        //   SceneController.getInstance().refresh();
    }

}
```

6.4 Control.Network

6.4.1 GameClient

Listing 19: GameClient.java

```
package Control.Network;

import Control.Controllers.GameController;
import Control.Network.Packets.Packet;
import java.io.IOException;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * A Game Client is responsible for client side communication
 * with the server in a network-multiplayer game.
 */
public class GameClient
{

    /**
     * IP-Address on which to connect to the Game's Server
     */
    private String _hostname = null;

    /**
     * Port on which to connect to the Game's Server
     */
    private int _port = 0;

    /**
     * playerNumber of the player on this client.
     */
    private int _playerNumber = 0;

    /**
     * if this client's game setup is ready.
     */
    private boolean _ready = false;

    /**
     * This Client's send thread, responsible for sending packets to the server.
     */
    private SendThread _sender = null;

    /**
```

```
43      * This Client's receive thread, responsible for receiving packets incoming
44      * from the server.
45      */
46     private RecieveThread _reciever = null;

48     /**
49      * Getter.
50      *
51      * @return True if the client has pressed the readyButton.
52      */
53     public boolean isReady ()
54     {
55         return this._ready;
56     }

58     /**
59      * Setter.
60      *
61      * @param ready True if the client has pressed the readyButton.
62      */
63     public void setReady (boolean ready)
64     {
65         this._ready = ready;
66     }

68     /**
69      * Getter.
70      *
71      * @return The PlayerNumber of this client.
72      */
73     public int getPlayerNumber ()
74     {
75         return _playerNumber;
76     }

78     /**
79      * Setter.
80      *
81      * @param playerNumber The desired PlayerNumber for this client.
82      */
83     public void setPlayerNumber (int playerNumber)
84     {
85         this._playerNumber = playerNumber;
86     }

88     /**
89      * Constructs a new Game Client.
90      *
91      * @param hostname The IP or Hostname this client wants to connect to.
92      * @param port     The port the server is open on.
93      */
94     public GameClient (String hostname, int port)
95     {
96         this._port = port;
97         this._hostname = hostname;
98     }

100    /**
101     * sets up communication with the server
102     */
103    public void execute ()
104    {
105        try
106          {
107            //Connect socket to server
108            Socket socket = new Socket(_hostname, _port);
```

```
            //Create and start communication threads.
            _sender = new SendThread(socket, this);
            _sender.start();
            _reciever = new RecieveThread(socket, this);
            _reciever.start();
          }
        catch (IOException ex)
          {
            Logger.getLogger(GameClient.class.getName())
                                                .log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Sends a packet p to the UserThread on the server that handles this
     * client.
     *
     * @param p The Package that is to be sent.
     */
    public void sendPackage (Packet p)
    {
        if (GameController.getInstance().getCurrentGame().isMultiplayer())
          {
            this._sender.send(p);
            return;
          }
    }

    /**
     * Interrupts both threads responsible for communication with the server.
     */
    public void shutDown ()
    {
        this._sender.interrupt();
        this._reciever.interrupt();
    }

}
```

6.4.2 GameServer

Listing 20: GameServer.java

```
package Control.Network;

import Control.Network.Packets.Packet;
import Resource.Constants.GeneralConstants.ControllerConstants;
import static Resource.Constants.GeneralConstants.NetworkConstants.*;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

/**
 * The game Server keeps track of all connected clients. For each connected
 * client exists one user thread that manages interactions with that client.
 */
public class GameServer extends Thread
{

    /**
     * The port on which a connection to the server can be made.
     */
    private int _port = DEFAULT_PORT;
```

```
    /**
     * List of all user threads.
     */
    private ArrayList<UserThread> _userThreads = new ArrayList<>();

    /**
     * wether or not the game has ended by either player winning.
     */
    private boolean _gameEnded = false;

    /**
     * Constructs a new GameServer.
     *
     * @param port The Port this server will be listening on.
     */
    public GameServer (int port)
    {
        this._port = port;
        this.setName(GAME_SERVER_THREAD_NAME);
    }

    /**
     * Allows for NUMBER_OF_PLAYERS connections to be made to this server. For
     * each connection creates one userThread that is responsible for that
     * connected client. Once the maximum amount of connections is made, alerts
     * all clients that the lobby is full.
     */
    public void run ()
    {
        try ( ServerSocket serverSocket = new ServerSocket(_port))
          {//serverSocket now waiting for incoming connections


            while (!this.isInterrupted() &&
                    _userThreads.size() < ControllerConstants.NUMBER_OF_PLAYERS)
              {//allows for up to NUMBER_OF_PLAYERS connections.

                //accept returns a new socket, that is connected to
                //whatever initiated the connection on the other end.
                Socket socket = serverSocket.accept();

                //new user Thread
                UserThread newUser = new UserThread(socket, this);
                _userThreads.add(newUser);
                newUser.start();

              }
            for (int i = 0; i < this._userThreads.size(); i++)
              {
                this.sendPackage(new Packet(PLAYER_NUMBER_FOR_PACKET + i),
                                                    this._userThreads.get(i));
              }
            this.sendPackage(new Packet(LOBBY_FULL));
          }
        catch (IOException ex)
          {
            ex.printStackTrace();
          }
    }

    /**
     * Sends a packet p to all clients via their user threads, except the user
     * that sent the packet.
     *
     * @param p      The package that is to be sent.
     * @param sender The UserThread sending the package.
```

```
     */
    public void sendPackage (Packet p, UserThread sender)
    {

        for (UserThread aUser : _userThreads)
          {
            if (aUser != sender)
              {
                aUser.sendPacket(p);
              }
          }
    }

    /**
     * Sends a packet p to all clients via their user threads.
     *
     * @param p the package that is to be sent.
     */
    public void sendPackage (Packet p)
    {

        for (UserThread aUser : _userThreads)
          {

            aUser.sendPacket(p);

          }
    }

    /**
     * Getter.
     *
     * @return if the game ended or not.
     */
    public boolean isGameEnded ()
    {
        return _gameEnded;
    }

    /**
     * Setter
     *
     * @param gameEnded True if the game has ended.
     */
    public void setGameEnded (boolean gameEnded)
    {
        this._gameEnded = gameEnded;
    }

    /**
     * Interrupts all user threads, then the server thread.
     */
    public void shutDown ()
    {
        for (UserThread u : this._userThreads)
          {
            u.interrupt();
          }
        this.interrupt();
    }

}
```

6.4.3 RecieveThread

Listing 21: RecieveThread.java

```
package Control.Network;

import Control.Controllers.SceneController;
import Control.Network.Packets.PacketDecoder;
import static Resource.Constants.GeneralConstants.NetworkConstants.EMPTY;
import static Resource.Constants.GeneralConstants.NetworkConstants.RECIEVE_THREAD_NAME;
import View.PopUps.ConnectionPopUp;
import java.io.IOException;
import java.io.InputStream;
import java.net.Socket;
import java.net.SocketException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Platform;

/**
 *
 */
public class RecieveThread extends Thread
{
    /**
     * Stream for incoming packets from the server's user thread..
     */
    private InputStream _in;

    /**
     * Socket for communication with the server's user thread.
     */
    private Socket _socket;

    /**
     * Client this Receive Thread receives for.
     */
    private GameClient _client;

    /**
     * Creates a new ReceiveThread.
     *
     * @param socket socket that is connected to the server.
     * @param client this receiver is created by
     */
    public RecieveThread (Socket socket, GameClient client)
    {
        this._client = client;
        this._socket = socket;

        this.setName(RECIEVE_THREAD_NAME
                + this._socket.getInetAddress().getHostAddress().toString());

        try
          {
            this._in = socket.getInputStream();
          }
        catch (IOException ex)
          {
            Logger.getLogger(RecieveThread.class.getName()).log
                (Level.SEVERE, null, ex);
          }
    }

    /**
     * While the thread runs, it receives incoming packets.
     */
    public void run ()
    {
```

```
        //to only display an error message on lost connection once.
        boolean connectionInterrupted = false;

        while (!this.isInterrupted())
          {
            byte[] input = null;
            try
              {//Read from stream
                input = new byte[1024];
                this._in.read(input);
                String inputstring = EMPTY;
                for (byte b : input)
                  {
                    inputstring += b;
                  }
              }
            catch (SocketException e)
              {//If connection is ost, inform the user by a popUp.
                if (connectionInterrupted == false)
                  {
                    Platform.runLater(() ->
                      {
                        new ConnectionPopUp().display();
                      });
                    connectionInterrupted = true;
                  }
              }
            catch (IOException ex)
              {
                Logger.getLogger(RecieveThread.class.getName()).log
                    (Level.SEVERE, null, ex);
              }

            if (input != null && input[0] != 0)
              {//if something useful was received, decode the packet.
                PacketDecoder.getInstance().DecodePacket(
                        (PacketDecoder.getInstance().fromByteArrayToPacket(input))
                );
                Platform.runLater(() ->
                  {
                    SceneController.getInstance().refresh();
                  });
              }

          }
    }

}
```

6.4.4 SendThread

Listing 22: SendThread.java

```
package Control.Network;

import Control.Network.Packets.Packet;
import static Resource.Constants.GeneralConstants.NetworkConstants.SEND_THREAD_NAME;
import static Resource.Constants.GeneralConstants.NetworkConstants.SEND_THREAD_SLEEP_TIME;
import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import sun.misc.Queue;

/**
```

```
14  *
15  */
16 public class SendThread extends Thread
17 {
18     /**
19      * Stream for outgoing packets to the server's user thread.
20      */
21     private OutputStream _out = null;

23     /**
24      * Socket for communication with the server's user thread.
25      */
26     private Socket _socket = null;

28     /**
29      * Client this send Thread sends for.
30      */
31     private GameClient _client = null;

33     /**
34      * queue that stores packets for sending.
35      */
36     private Queue _queue = new Queue();

38     /**
39      * packet that is currently ready for sending.
40      */
41     private Packet _sendPacket = null;

43     /**
44      * Creates a new Send Thread.
45      *
46      * @param socket socket that is connected to the server.
47      * @param client this sender is created by
48      */
49     public SendThread (Socket socket, GameClient client)
50     {
51         this._client = client;
52         this._socket = socket;

54         this.setName(SEND_THREAD_NAME +
55                  this._socket.getInetAddress().getHostAddress().toString());


58         try
59           {
60             this._out = socket.getOutputStream();
61           }
62         catch (IOException ex)
63           {
64             Logger.getLogger(SendThread.class.getName())
65                                                 .log(Level.SEVERE, null, ex);
66           }
67     }

69     /**
70      * Adds a packet to the queue for sending.
71      *
72      * @param p the package that is to be sent.
73      */
74     public void send (Packet p)
75     {
76         _queue.enqueue(p);
77     }

79     /**
```

```
     * While the thread runs, it regularly checks for packets to send, then
     * sends them.
     */
    public void run ()
    {
        while (!this.isInterrupted())
          {
            if (!_queue.isEmpty())
              {//If there is anything to send...
                try
                  {//...send it.
                    Packet send = (Packet) _queue.dequeue();
                    this._out.write(send.toByteArray());
                  }
                catch (IOException | InterruptedException ex)
                  {
                    Logger.getLogger(SendThread.class.getName())
                                                    .log(Level.SEVERE, null, ex);
                  }
              }
            else
              {//If there is nothing to send, sleep.
                try
                  {
                    this.sleep(SEND_THREAD_SLEEP_TIME);
                  }
                catch (InterruptedException ex)
                  {
                    Logger.getLogger(SendThread.class.getName())
                                                    .log(Level.SEVERE, null, ex);
                  }
              }
          }
    }

}
```

6.4.5 UserThread

Listing 23: UserThread.java

```
package Control.Network;

import Control.Network.Packets.Packet;
import Control.Network.Packets.PacketDecoder;
import static Resource.Constants.GeneralConstants.NetworkConstants.PACKET_SIZE;
import static Resource.Constants.GeneralConstants.NetworkConstants.USER_THREAD_NAME;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * For each client connected to a server a userThread is created. The user
 * thread is responsible for incoming and outdoing communication to that client.
 */
public class UserThread extends Thread
{

    /**
     * Socket this user thread communicates by.
     */
    private Socket _socket = null;
```

```
26      /**
27       * Server this user thread was created by.
28       */
29      private GameServer _server = null;

31      /**
32       * output stream to send packets out to.
33       */
34      private OutputStream _out = null;

36      /**
37       * For each client connected to a server a userThread is created. The user
38       * thread is responsible for incoming and outdoing communication to that
39       * client.
40       *
41       * @param socket this userThreads socket.
42       * @param server the server this user wants to connect to.
43       */
44      public UserThread (Socket socket, GameServer server)
45      {
46          this._socket = socket;
47          this._server = server;
48          this.setName(USER_THREAD_NAME +
49                  this._socket.getInetAddress().getHostAddress().toString());
50      }

52      /**
53       * regularly checks for incoming packets, while running.
54       */
55      @Override
56      public void run ()
57      {
58          InputStream input = null;
59          OutputStream output = null;

61          try
62            {
63              input = _socket.getInputStream();
64              output = _socket.getOutputStream();
65              this._out = output;

67              do
68                /*
69                 * Loop that handles all packets incoming from the client
70                 * this user thread is responsible for.
71                 * All incoming packets are sent to every user thread by the
72                 * server,
73                 * except this one, for passing them on to the corresponding
74                 * client.
75                 */
76                {
77                  byte[] in = new byte[PACKET_SIZE];
78                  input.read(in);
79                  Packet p = PacketDecoder.getInstance().fromByteArrayToPacket(in);

81                  _server.sendPackage(p, this);
82                }
83              while (!this.isInterrupted() && !_server.isGameEnded());
84            }
85          catch (IOException ex)
86            {
87              Logger.getLogger(UserThread.class.getName())
88                                                  .log(Level.SEVERE, null, ex);
89            }
90          /*
91           * Once the game ended or the thread is interrupted, close all streams
```

```
         */
        finally
          {
            try
              {
                input.close();
                output.close();
              }
            catch (IOException | NullPointerException ex)
              {
                Logger.getLogger(UserThread.class.getName())
                                                .log(Level.SEVERE, null, ex);
              }
          }
    }

    /**
     * Sends a packet p to the client this user thread is connected to.
     *
     * @param p the package that is to be sent.
     */
    public void sendPacket (Packet p)
    {
        try
          {
            this._out.write(p.toByteArray());
          }
        catch (IOException ex)
          {
            Logger.getLogger(UserThread.class.getName())
                                                .log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Getter
     *
     * @return this user thread's socket.
     */
    public Socket getSocket ()
    {
        return this._socket;
    }

}
```

6.5 Control.Networks.Packets

6.5.1 Packet

Listing 24: Packet.java

```
package Control.Network.Packets;

import static Resource.Constants.GeneralConstants.ControllerConstants.SPACE;
import static Resource.Constants.GeneralConstants.NetworkConstants.*;

/**
 * The Packet class packs all information about the players turns in the game
 * so it can be send to the opponent player.
 */
public class Packet
{
    /**
     * The content that is stored in this packet.
```

```
14      */
15     private char[] _content = null;

17     /**
18      * Type of this Packet, empty by default.
19      */
20     private PacketType _type = null;

22     /**
23      * Default Constructor.
24      */
25     public Packet ()
26     {
27         this._content = new char[PACKET_SIZE];
28         this._type = PacketType.EMPTY;
29     }

31     /**
32      * Constructor that takes a String as Packet content.
33      *
34      * @param content - the String that has to be send with the packet.
35      */
36     public Packet (String content)
37     {
38         this._content = new char[PACKET_SIZE];
39         this._type = PacketType.EMPTY;
40         if (content.contains(SPACE))
41           {
42             this.evaluateType(content.split(SPACE)[0]);
43           }
44         else
45           {
46             this.evaluateType(content);
47           }

49         if (content.length() <= PACKET_SIZE && this._type != PacketType.INVALID)
50           {
51             for (int i = 0; i < content.toCharArray().length; i++)
52               {
53                 this._content[i] = content.charAt(i);
54               }
55           }
56         else
57           {
58             System.err.println(PACKET_IS_INVALID);
59           }
60     }

62     /**
63      * Constructor that takes a char array as content.
64      *
65      * @param content - the content that has to be send with the packet.
66      */
67     public Packet (char[] content)
68     {
69         this.evaluateType(charArrayToString(content).split(SPACE)[0]);

71         if (content.length <= PACKET_SIZE && this._type != PacketType.INVALID)
72           {
73             this._content = content;
74           }
75     }

77     /**
78      * Getter for Packets content
79      *
```

```
     * @return - a char array with the content of the packet.
     */
    public char[] getContent ()
    {
        return this._content;
    }

    /**
     * getter for the PacketType of this Packet.
     *
     * @return - this Packets PacketType.
     */
    public PacketType getType ()
    {
        return this._type;
    }

    /**
     * evaluates the type of the Packet with a given String.
     *
     * @param type - type that has to be evaluated
     */
    private void evaluateType (String type)
    {
        switch (type)
          {
            case "playcard":
                this._type = PacketType.PLAYCARD;
                break;
            case "endturn":
                this._type = PacketType.ENDTURN;
                break;
            case "move":
                this._type = PacketType.MOVE;
                break;
            case "test":
                this._type = PacketType.TEST;
                break;
            case "lobby":
                this._type = PacketType.LOBBY;
                break;
            case "playernumber":
                this._type = PacketType.PLAYERNUMBER;
                break;
            case "hand":
                this._type = PacketType.HAND;
                break;
            default:
                this._type = PacketType.INVALID;
                break;
          }
    }

    /**
     * Process a given byte array to a string.
     *
     * @param input - the char array that has to become a string.
     *
     * @return - the char array as a String.
     */
    private String charArrayToString (char[] input)
    {
        String result = EMPTY;
        for (char c : input)
          {
            result += c;
```

```
            }
        return result;
    }

    /**
     * Getter for the content of this Packet as a byte array.
     *
     * @return - content of this packet as a byte array.
     */
    public byte[] toByteArray ()
    {
        byte[] bytes = new String(this._content).getBytes();

        return bytes;
    }

    /**
     * toString gives you all necessary information of this Packet in one
     * String.
     *
     * @return - the String that represents the data of this packet.
     */
    @Override
    public String toString ()
    {
        String result = EMPTY;

        for (char c : this._content)
          {
            result += c;
          }

        return result;
    }
}
```

6.5.2 PacketDecoder

Listing 25: PacketDecoder.java

```
package Control.Network.Packets;

import Control.Controllers.*;
import Model.Card.AllCards.CardFactory;
import Model.Map.Map;
import Model.Map.Tile;
import Model.Player;
import static Resource.Constants.GeneralConstants.ControllerConstants.SPACE;
import static Resource.Constants.GeneralConstants.NetworkConstants.*;
import javafx.application.Platform;

/**
 * The PacketDecoder decodes the received packets to perform the submitted
 * actions of the opponent Player.
 */
public class PacketDecoder
{

    /**
     * The Instance to make the PcketDecoder a singleton.
     */
    private static final PacketDecoder _instance = new PacketDecoder();

    /**
     * Getter for the instance of the PacketDecoder
     *
```

```
27      * @return - PacketDecoder instance
28      */
29     public static PacketDecoder getInstance ()
30     {
31         return _instance;
32     }

34     /**
35      * DecodePacket determines the information of a given Packet and invokes the
36      * submitted actions of the opponent player.
37      *
38      * @param packet - the Packet that has to be decoded.
39      */
40     public void DecodePacket (Packet packet)
41     {
42         String[] information = new String[4];
43         int[] intinf = new int[4];

45         //Removes the first Word of the Packet.
46         for (int i = 0; i < packet.toString().split(SPACE).length; i++)
47           {
48             if (i != 0)
49               {
50                 information[i - 1] = packet.toString().split(SPACE)[i];
51                 try
52                   {
53                     intinf[i - 1] = Integer.parseInt
54                         (this.removeUselessCharacters(information[i - 1]));
55                   }
56                 catch (NumberFormatException e)
57                   {
58                     //information is not a number
59                   }
60               }
61           }

63         processPacket(packet, information, intinf);
64     }

66     /***
67      * Determines what type a packet is. then invokes another method to
68      * process what the packet makes happen.
69      *
70      * @param packet packet to process
71      * @param information
72      * @param intinf
73      */
74     private void processPacket (Packet packet, String[] information, int[] intinf)
75     {
76         switch (packet.getType())
77           {
78             case MOVE:
79                 processMovePacket(intinf);
80                 break;

82             case PLAYCARD:
83                 processCardPacket(intinf);
84                 break;

86             case LOBBY:
87                 processLobbyPacket(information[0]);
88                 break;

90             case HAND:
91                 processHandPacket(intinf);
92                 break;
```

```
            case PLAYERNUMBER:
                GameController.getInstance().getClient().setPlayerNumber(intinf[0]);
                break;

            case TEST:
                break;

            case ENDTURN:
                TurnController.getInstance().nextPlayer();
                break;

            case INVALID:
                System.err.println(ABORTING);
                break;

            default:
                System.err.println(ABORTING);
                break;
        }
    }

    /**
     * If a movePacket was received, executes that move (or possibly attack).
     * @param var
     */
    private void processMovePacket (int[] var)
    {
        Map map = MapController.getInstance().getMap();

        //var 0 and 1 are the coordinates of the first selected tile.
        //var 2 and 3 are the coordinates of the second selected tile.
        InputController.getInstance().processTileSelection
        (map.getTile(var[0], var[1]), map.getTile(var[2], var[3]), false);
    }

    /**
     * Processes different game setup packets.
     * @param inf
     */
    private void processLobbyPacket (String inf)
    {
        switch (this.removeUselessCharacters(inf))
          {
            case "full":
                SceneController.getInstance().setSceneToGameSetupSceneMP();
                GameController.getInstance().startGame();
                break;
            case "ready":
                if (GameController.getInstance().getClient().isReady())
                  {
                    GameController.getInstance().getClient().sendPackage
                        (new Packet(LOBBY_STARTGAME));
                    setGameScene();
                  }
                break;
            case "startgame":
                SoundController.getInstance().getMusic().stopMenuMusic();
                SoundController.getInstance().getMusic().playStartGame();
                setGameScene();
                break;
          }
    }

    /**
     * Activates the game scene.
```

```
     */
    private void setGameScene ()
    {
        Platform.runLater(() ->
          {
            SceneController.getInstance().setSceneToGameScene();
          });
    }

    /**
     * processes a hand packet.
     * @param var
     */
    private void processHandPacket (int[] var)
    {
        Player opponent;

        if (GameController.getInstance().getClient().getPlayerNumber() == 0)
          {
            opponent = PlayerController.getInstance().getPlayer(1);
          }
        else
          {
            opponent = PlayerController.getInstance().getPlayer(0);
          }

        if (var[0] < 0)
          {
            if (var[1] < 0)
              {
                for (int i = 0; i < (var[1] * (-1)); i++)
                  {
                    opponent.getHand().removeCard(0);
                  }
              }
            else
              {
                GameController.getInstance().getCurrentPlayerController()
                        .getPlayer(var[2]).getHand().removeCard(var[1]);
              }

          }
        else
          {
            opponent.getHand().addCard(CardFactory.getNewInstance(var[0]));
          }
    }

    /**
     * processes a card packet.
     * @param var
     */
    private void processCardPacket (int[] var)
    {
        Map map = MapController.getInstance().getMap();
        Tile target = map.getTile(var[3], var[2]);
        GameController.getInstance().getCurrentPlayerController()
                .getPlayer(var[1]).playCard(var[0], target);
    }

    /**
     * fromByteArrayToPacket turns a byte array into a new Packet.
     *
     * @param data - the byte array that has to be processed
     *
     * @return - a new packet that contains the data of the given byte array
```

```
     */
    public Packet fromByteArrayToPacket (byte[] data)
    {
        return new Packet(removeUselessCharacters(new String(data)));
    }

    /**
     * removeUselessCharacters removes all Characters from a given string that
     * are useless for packet decoding.
     *
     * @param s - the string where the chars has to be removed from
     *
     * @return - a clean string without useless chars
     */
    private String removeUselessCharacters (String s)
    {
        String match = USEFUL_CHARS;

        StringBuilder result = new StringBuilder();

        for (char c : s.toCharArray())
          {
            if (match.contains(c + EMPTY))
              {
                result.append(c);
              }

          }
        return result.toString();
    }

}
```

6.5.3 PacketType

Listing 26: PacketType.java

```
package Control.Network.Packets;

/**
 * The Packet types that are available to be sent.
 */
public enum PacketType
{

    /**
     * a movePacket contains two sets of tile-coordinates
     * that, when processed, are a move, an attack move or an attack.
     */
    MOVE,

    /**
     * a playCardPacket contains a cardID, the ID of the payer who plays it and
     * the coordinates of the target tile.
     */
    PLAYCARD,
    /**
     * an endTurnPacket is sent, whenever a player ends their turn, indicating
     * it is the next players turn.
     */
    ENDTURN,
    /**
     * LobbyPackets are used to communicate when the lobby is full, when
     * a player is ready to start and when to start the game.
     */
    LOBBY,
```

```
    /**
     * TestPackets are used to test whether our packet system works. They will
     * be no longer used upon finishing this project.
     */
    TEST,
    /**
     * Empty Packets do not contain any information.
     */
    EMPTY,
    /**
     * PlayerNumber Packets are sent by the server to tell all players which
     * number they are.
     */
    PLAYERNUMBER,
    /**
     * an invalid packet contains information that can't be used.
     */
    INVALID,
    /**
     * HandPackets are sent to tell the other player that a card has been added
     * or removed from a players hand.
     */
    HAND
}
```

6.6 Control.SemiControllers

6.6.1 Highlighter

Listing 27: Highlighter.java

```
package Control.SemiControllers;

import Control.Controllers.GameController;
import Control.Controllers.MapController;
import Model.GameObject.Attribute;
import Model.GameObject.CreatureAdapter;
import Model.Map.Pathfinding;
import Model.Map.Tile;
import java.util.ArrayList;

/**
 * The Highlighter is a Singleton that could determine the possible moves of
 * gameobject on a given Tile
 *
 * @author Henning
 */
public class Highlighter
{

    /**
     * Instance for making the Highlighter a Singleton
     */
    private static Highlighter _instance = new Highlighter();

    /**
     * List of tiles that have been highlighted by the highlightpossibleMoves
     * method, for unhighlighting
     */
    private ArrayList<Tile> _highlightedMoveTiles = new ArrayList<Tile>();

    /**
     * List of tiles that have been highlighted by the highlightPossibleAttacks
     * method, for unhighlighting
     */
```

```
35      private ArrayList<Tile> _highlightedAttackTiles = new ArrayList<Tile>();

37      /**
38       * Getter for the _instance of the singleton MoevementController
39       *
40       * @return - the Highlighter _instance
41       */
42      public static Highlighter getInstance ()
43      {
44          return _instance;
45      }

47      private ArrayList<Tile> determinePossibleAttacks (CreatureAdapter c)
48      {
49          ArrayList<Tile> results = new ArrayList<>();

51          //this probably is rather inefficient but... whatever it takes, right?
52          for (Tile tile : this._highlightedMoveTiles)
53            {
54              for (Tile t : MapController.getInstance().getMap()
55                      .getTilesInRange(tile, c.getAttackRange()))
56                {
57                  if (CreatureAdapter.class.isInstance(t.getGameObject()))
58                    {
59                      CreatureAdapter target = (CreatureAdapter) t.getGameObject();
60                      if (target.getOwner() != GameController.getInstance()
61                              .getCurrentTurnController().getCurrentPlayer()
62                              .getOwnerNumber() && MapController.getInstance()
63                                      .getMap().lineOfSight(c, target))
64                        {
65                          results.add(t);
66                        }
67                    }
68                }
69            }

71          return results;
72      }

74      /**
75       * Determines the possible tiles where the object on the given tile could
76       * move
77       *
78       * @param c - the Creature on the tile from where the possible moves should
79       *           be determined
80       *
81       * @return - an ArrayList with all possible Tiles to move to
82       */
83      private ArrayList<Tile> determinePossibleMoves (CreatureAdapter c)
84      {
85          Tile t = c.getTile();

87          ArrayList<Tile> results = new ArrayList<>();

89          if (!c.hasAttribute(Attribute.Paralyzed))
90            {
91              if (!(c.hasAttacked()))
92                {
93                  for (Tile tile : MapController.getInstance().getMap()
94                          .getTilesInRange(t, (c.getMovementpoints())))
95                    {
96                      if ((Pathfinding.getInstance().calcMovementPoints(t, tile,
97                      false, (CreatureAdapter) t.getGameObject())) <=
98                      ((CreatureAdapter) t.getGameObject()).getMovementpoints())
99                        {
100                         results.add(tile);
```

```
                        }
                    }
                }
            }
        return results;
    }

    /**
     * unhighlights all tiles highlighted for moves or actions
     */
    public void unhighlight ()
    {
        if (this._highlightedMoveTiles != null && !this._highlightedMoveTiles
                .isEmpty())
            {
              for (Tile t : this._highlightedMoveTiles)
                {
                  t.setIsHighlightedForMove(false);
                }
              this._highlightedMoveTiles.clear();
            }
        if (this._highlightedAttackTiles != null && !this._highlightedAttackTiles
                .isEmpty())
            {
              for (Tile t : this._highlightedAttackTiles)
                {
                  t.setIsHighlightedForAttack(false);
                }
              this._highlightedAttackTiles.clear();
            }
    }

    /**
     * Highlights a tile in a color indicating that it can be targeted by the
     * selected spell.
     *
     * @param t the tile to be highlighted.
     */
    public void highlightSpellTarget (Tile t)
    {
        this._highlightedAttackTiles.add(t);
        t.setIsHighlightedForAttack(true);

    }

    /**
     * Highlights multiple tiles in a color indicating that they can be targeted
     * by the selected spell.
     *
     * @param targets the tiles to be highlighted.
     */
    public void highlightSpellTarget (ArrayList<Tile> targets)
    {
        this._highlightedAttackTiles.addAll(targets);

        for (Tile tiles : this._highlightedAttackTiles)
            {
              tiles.setIsHighlightedForAttack(true);
            }
    }

    /**
     * Highlights all tiles a Creature is able to attack.
     *
     * @param c the selected Creature.
     */
```

```
    public void highlightPossibleAttacks (CreatureAdapter c)
    {
        this._highlightedAttackTiles = determinePossibleAttacks(c);
        for (Tile tiles : this._highlightedAttackTiles)
          {
            tiles.setIsHighlightedForAttack(true);
          }

    }

    /**
     * Highlights all tiles where a given Creature is able to move to with its
     * remaining movementpoints.
     *
     * @param c The moving Creature.
     */
    public void highlightPossibleMoves (CreatureAdapter c)
    {
        this._highlightedMoveTiles = determinePossibleMoves(c);
        for (Tile tiles : this._highlightedMoveTiles)
          {
            tiles.setIsHighlightedForMove(true);
          }
    }

}
```

6.6.2 Selector

Listing 28: Selector.java

```
package Control.SemiControllers;

import Control.Controllers.GameController;
import Control.Controllers.InputController;
import Control.Controllers.PlayerController;
import Control.Network.Packets.Packet;
import Control.Network.Packets.PacketType;
import Exceptions.UnexpectedCardException;
import Model.Card.ICard;
import Model.GameObject.CreatureAdapter;
import Model.Map.Tile;
import static Resource.Constants.GeneralConstants.ControllerConstants.SPACE;
import java.util.logging.Level;
import java.util.logging.Logger;

/*
 * The selector stores and selects a primary and secondary tile.
 * these will later be determined by left or right clicking on the map.
 * whenever either of them changes, the processTileSelection() method of the
 * inputController is called.
 */
public class Selector
{

    private static Selector _instance = new Selector();

    /**
     * currently selected Card
     */
    private ICard _selectedCard = null;

    /**
     * index of a card, meaning its position in the players hand.
     */
    private int _selectedCardIndex = -1;
```

```
37      /**
38       * primary selected Tile aka. Tile that has been left-clicked on
39       *
40       */
41      private Tile _primSelectedTile = null;

43      /**
44       * secondary selected Tile aka. Tile that has been right-clicked on
45       *
46       */
47      private Tile _secSelectedTile = null;

49      /**
50       * Getter.
51       *
52       * @return The Index in the players hand for the currently selected card.
53       */
54      public int getSelectedCardIndex ()
55      {
56          return _selectedCardIndex;
57      }

59      /**
60       * Singleton Getter.
61       *
62       * @return The Instance of this Singleton.
63       */
64      public static Selector getInstance ()
65      {
66          return _instance;
67      }

69      /**
70       * Getter.
71       *
72       * @return The currently selected card.
73       */
74      public ICard getSelectedCard ()
75      {
76          return _selectedCard;
77      }

79      /**
80       * Getter.
81       *
82       * @return The currently selected first tile.
83       */
84      public Tile getPrimSelectedTile ()
85      {
86          return this._primSelectedTile;
87      }

89      /**
90       * Getter.
91       *
92       * @return The currently selected second tile.
93       */
94      public Tile getSecSelectedTile ()
95      {
96          return this._secSelectedTile;
97      }

99      /**
100      * Setter.
101      *
```

```
     * @param t the current secondary selected tile.
     */
    public void setSecSelectedTile (Tile t)
    {
        this._secSelectedTile = t;
    }

    /**
     * Hides the tooltip for the currently selected card.
     */
    private void hideCardTooltip ()
    {
        if (this._selectedCard != null)
          {
            if (this._selectedCard.getTooltip() != null)
              {
                this._selectedCard.getTooltip().hide();
              }
          }
    }

    private void hideMinionTooltip ()
    {
        if (this._primSelectedTile != null)
          {
            if (this._primSelectedTile.getGameObject() != null)
              {
                if (this._primSelectedTile.getGameObject().getTooltip() != null)
                  {
                    this._primSelectedTile.getGameObject().getTooltip()
                            .hide(this._primSelectedTile.getGameObject().getTooltip());
                  }
              }
          }
    }

    /**
     * Unhighlights all highlights. then checks wether a card, primary or
     * secondary tile are selected and processes selection accordingly.
     */
    private void processSelection ()
    {
        Highlighter.getInstance().unhighlight();

        if (this._selectedCard != null)
          {
            if (this._primSelectedTile != null)
              {
                this.playCard();
                this.deselectCard();
              }
          }

        else
          {
            if (this._primSelectedTile == null)
              {
                if (this._secSelectedTile != null)
                  {
                    /*
                     * System.out.println("no primaryTile selected;
                     * secondary Tile set to null; no actions attempted");
                     */
                  }
                this._secSelectedTile = null;
              }
```

```
            else
              //primary tile is selected, secondary tile might not be selected
              {

                boolean creatureOnPrimary = false;
                creatureOnPrimary = InputController.getInstance()
                        .processPrimaryTileSelection(this._primSelectedTile);

                if (creatureOnPrimary == true)
                  {
                    //if there's a creature on primaryTile highlight moves & attacks
                    Highlighter.getInstance().highlightPossibleMoves(
                            (CreatureAdapter) this._primSelectedTile.getGameObject());
                    Highlighter.getInstance().highlightPossibleAttacks(
                            (CreatureAdapter) this._primSelectedTile.getGameObject();

                    if (this._secSelectedTile != null)
                      {
                        InputController.getInstance().processTileSelection(
                                this._primSelectedTile, this._secSelectedTile,
                                GameController.getInstance().getCurrentTurnController()
                                        .getCurrentPlayer().getOwnerNumber());
                      }
                    else
                      {
                        /*
                         * System.out.println("primaryTile selected;
                         * no secondary Tile selected; no actions attempted");
                         */
                      }
                  }
              }
          }
      }

    /**
     * selects primary Tile from Tile
     *
     * @param t The tile to be selected.
     */
    public void selectTile (Tile t)
    {
        if (_primSelectedTile != null)
          {
            deselectTile();
          }

        if (t.getGameObject() != null)
          {
            hideCardTooltip();
            t.getGameObject().getTooltip().show(t.getGameObject().getTooltip());
          }

        this._primSelectedTile = t;
        t.setIsSelected(true);

        this.processSelection();
    }

    /**
     * selects secondary Tile from String read from textField
     *
     * @param input the secondary tile to be selected.
     */
    public void selectSecTile (Tile input)
    {
```

```
        this._secSelectedTile = input;
        this.processSelection();
    }

    /**
     * Selects a Card if the input string points to a card in the current
     * players hand.
     *
     * @param c input the input string
     */
    public void selectCard (ICard c)
    {

        if (c != null)
          {
            hideCardTooltip();
            this._selectedCard = c;
            if (this._selectedCard.getTooltip() != null)
              {
                hideMinionTooltip();
                this._selectedCard.getTooltip().show();
              }
            this._selectedCardIndex = GameController.getInstance()
                    .getCurrentTurnController().getCurrentPlayer().getHand()
                    .getIndexOf(c);
//              System.out.println("You selected: " + this._selectedCard);
          }

        try
          {
            InputController.getInstance().processCardSelection();
          }
        catch (UnexpectedCardException e)
          {
            Logger.getLogger(PlayerController.class.getName())
                    .log(Level.SEVERE, null, e);
          }
    }

    /**
     * This function plays the currently selected card.
     */
    public void playCard ()
    {
        if (this._selectedCard == null || this._selectedCardIndex == -1)
          {
//              System.out.println("Please select a card first.");
            return;
          }
        if (GameController.getInstance().getCurrentGame().isMultiplayer())
          {
            GameController.getInstance().getClient().sendPackage(
                    new Packet(PacketType.PLAYCARD.toString().toLowerCase() +
                             SPACE + _selectedCardIndex + SPACE +
                            GameController.getInstance().getCurrentTurnController()
                                    .getCurrentPlayer().getOwnerNumber() + SPACE +
                            this._primSelectedTile.getAxialTransform().getD() +
                            SPACE + this._primSelectedTile.getAxialTransform()
                                    .getH()));
          }
        GameController.getInstance().getCurrentTurnController().getCurrentPlayer()
                .playCard(this._selectedCardIndex, this._primSelectedTile);
    }

    /**
     * Deselects the currently selected card. This function has to be called
```

```
     * when the turn is ended.
     */
    public void deselectCard ()
    {
        if (this._selectedCard != null)
          {
            hideCardTooltip();
            this._selectedCard = null;
            this._selectedCardIndex = -1;
          }
    }

    /**
     * deselects primary selected tile
     */
    public void deselectTile ()
    {
        if (this._primSelectedTile != null)
          {
            this._primSelectedTile.setIsSelected(false);

            hideMinionTooltip();

            this._primSelectedTile = null;
          }
        this.processSelection();
    }

}
```

6.7 Exceptions

6.7.1 MapInvalidException

Listing 29: MapInvalidException.java

```
package Exceptions;

/**
 *
 */
public class MapInvalidException extends RuntimeException
{

}
```

6.7.2 MapTypeNotSupportedException

Listing 30: MapTypeNotSupportedException.java

```
package Exceptions;

/**
 *
 */
public class MapTypeNotSupportedException extends Exception
{

}
```

6.7.3 UnexpectedCardException

Listing 31: UnexpectedCardException.java

```
package Exceptions;

/**
 *
 */
public class UnexpectedCardException extends Exception
{

}
```

6.7.4 UnexpectedGameObjectException

Listing 32: .java

```
package Exceptions;

/**
 *
 */
public class UnexpectedGameObjectException extends RuntimeException
{

}
```

6.8 Model

6.8.1 Player

Listing 33: Player.java

```
package Model;

import Control.Controllers.GameController;
import Control.Network.Packets.Packet;
import Model.Card.Deck;
import Model.Card.Hand;
import Model.Card.Wand;
import Model.GameObject.CreatureAdapter;
import Model.GameObject.PlayerCharacter;
import Model.Map.Tile;
import static Resource.Constants.GeneralConstants.NetworkConstants.HAND_MINUS_ONE_MINUS;
import Resource.Constants.GeneralConstants.PlayerConstants;
import java.io.Serializable;
import java.util.Vector;

/**
 * The Player Class that represents the player at the keyboard with all his
 * values.
 */
public class Player implements Serializable
{

    /**
     * the ownerNumber of the owning Player.
     */
    private int _ownerNumber = 0;

    /**
     * the Mana value of the player.
     */
```

```
31      private int _mana = 0;

33      /**
34       * the name of the Player.
35       */
36      private String _name = null;

38      /**
39       * the playerCharecter of the Wizard for the Player.
40       */
41      private PlayerCharacter _character = null;

43      /**
44       * true if it is this Players turn.
45       */
46      private boolean _currentPlayer = false;

48      /**
49       * this is the Hand of the Player here we can no how many Cards they have.
50       */
51      private Hand _hand = null;

53      /**
54       * Stores all Creatures that are owned by this Player.
55       */
56      private Vector<CreatureAdapter> _myCreatures = null;

58      /**
59       * Constructor of player with parameter
60       *
61       * @param name        The Players Name.
62       * @param ownerNumber The Players Owner Number (ID)
63       */
64      public Player (String name, int ownerNumber)
65      {
66          this._hand = new Hand(this, new Deck());
67          this._name = name;
68          this._ownerNumber = ownerNumber;
69          this._mana = PlayerConstants.PLAYER_DEFAULT_MANA_AMOUNT;
70          this._character = new PlayerCharacter(name, ownerNumber);
71          this._myCreatures = new Vector<CreatureAdapter>();
72          this._myCreatures.addElement(this._character);
73      }

75      /**
76       * Getter for the Mana of this Player
77       *
78       * @return - the Mana value.
79       */
80      public int getMana ()
81      {
82          return this._mana;
83      }

85      /**
86       * reduces the players mana by the given amount.
87       *
88       * @param m the amount of mana reduction.
89       */
90      public void reduceMana (int m)
91      {
92          this._mana = (this._mana - m);
93      }

95      /**
96       * restores the players mana by the given amount.
```

```
97          *
98          * @param m the amount of mana restoration.
99          */
100         public void restoreMana (int m)
101         {
102             this._mana = (this._mana + m);
103         }

105         /**
106          * getter for the PlayerCharacter instance of this Player
107          *
108          * @return - the instance of this PlayerCharacter.
109          */
110         public PlayerCharacter getCharacterInstance ()
111         {
112             return this._character;
113         }

115         /**
116          * getter for ownerNumber of the Player.
117          *
118          * @return ownerNumber of the Player.
119          */
120         public int getOwnerNumber ()
121         {
122             return this._ownerNumber;
123         }

125         /**
126          * Getter for all the Creatures that are owned by this Player.
127          *
128          * @return - a Vector with all Creatures that are owned by this Player.
129          */
130         public Vector<CreatureAdapter> getMyCreatures ()
131         {
132             return this._myCreatures;
133         }

135         /**
136          * Adds a Creature to the Vector of Creatures that are owned by this Player.
137          *
138          * @param creature - creature that should be owned by this Player.
139          */
140         public void addCreature (CreatureAdapter creature)
141         {
142             this._myCreatures.addElement(creature);
143         }

145         /**
146          * use this to play a card with index to the target Tile.
147          *
148          * @param index  the card index
149          * @param target the target Tile
150          */
151         public void playCard (int index, Tile target)
152         {

154             if (this._mana < this._hand.getCard(index).getManaCost())
155               {
156                 return;
157               }

159             int manacost = -1;

161             manacost = this._hand.getCard(index).getManaCost();
```

```
        if (manacost < 0)
          {
            return;
          }

        if (this._hand.playCard(index, target))
          {
            this._mana -= manacost;
          }
    }

    /**
     * Plays a Card from the Players Hand of Cards.
     *
     * @param index - the Card on the Hand has an Index.
     */
    public void playCard (int index)
    {
        int manacost = -1;

        manacost = this._hand.getCard(index).getManaCost();

        if (manacost < 0)
          {
            return;
          }

        if (this._hand.playCard(index))
          {
            this._mana -= manacost;
          }
    }

    /**
     * Chooses a given wand for the player.
     *
     * @param w - the wand that was chosen.
     */
    public void chooseWand (Wand w)
    {
        if (this._hand.getSize() > 0)
          {
            GameController.getInstance().getClient().sendPackage(new Packet(HAND_MINUS_ONE_MINUS + t
          }
        this._hand = new Hand(this, new Deck(w));
    }

    /**
     * Getter for the Hand of the Player.
     *
     * @return - the hand of the Player
     */
    public Hand getHand ()
    {
        return this._hand;
    }

    /**
     * toString for GameObject.
     *
     * @return String that represents all data of the Player
     */
    @Override
    public String toString ()
    {
        StringBuilder sb = new StringBuilder();
```

```
        sb.append("Player").append(this._name).append('\n').append("Name: ")
                .append(this._name).append('\n').append("Size: ")
                .append(this._character.getSize()).append('\n')
                .append("Axialtransform: ").append(this._character.getTile()
                .toString()).append('\n');

        return sb.toString();
    }

}
```

6.9 Model.Card

6.9.1 CardAdapter

Listing 34: CardAdapter.java

```
package Model.Card;

import Control.Controllers.GameController;
import Control.Controllers.MapController;
import Model.Game.FluxMeter;
import Model.GameObject.GameObject;
import Model.Map.Tile;
import View.Tooltip.Tooltip;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.image.Image;

/**
 * The CardAdapter is the Blueprint for all kinds of Cards. all Playable cards
 * must extend this class. Cards can be used and then play an effect.
 */
public abstract class CardAdapter implements ICard
{

    /**
     * This Cards Name.
     * Mostly displayed in Tooltips.
     */
    private String _cardName = null;

    /**
     * The Amount of Mana it will cost to play this Card.
     */
    private int _manaCost = 0;

    /**
     * The Amount of Flux that will influence the Cosmic Flux Meter after this
     * Card is played.
     */
    private int _fluxChange = 0;

    /**
     * The Range from the Players PlayerCharacter this Cards Effect can be
     * executed in.
     *
     * Range = 0:
     * Can only be executed on the Player Character
     *
     * Range >=1:
     * Can be executed anywhere within a Radius around the PlayerCharacter.
```

```
48          */
49         private int castRange = 1;

51         /**
52          * The identity of this card.
53          */
54         private int id = 0;

56         /**
57          * The Effect that is to be Executed when this Card is played.
58          */
59         private ICardEffect _cardEffect = null;

61         /**
62          * This Cards Description (used for Tooltips).
63          */
64         private String _description = Resource.Constants.CardConstants.CardConstants
65                                                                .DEFAULT_CARD_DESCRIPTION;

67         /**
68          * This Cards Type (Law, Chaos or Neutral).
69          */
70         private CardType _type = null;

72         /**
73          * This Cards representing Image.
74          */
75         private Image _image = null;

77         /**
78          * This Cards Tooltip.
79          */
80         private Tooltip _tooltip = null;

82         /**
83          * Getter for the identity of this card.
84          *
85          * @return - id
86          */
87         @Override
88         public int getID ()
89         {
90             return this.id;
91         }

93         /**
94          * Setter for the ID of this card.
95          *
96          * @param id the desired ID for this card.
97          */
98         protected void setID (int id)
99         {
100            this.id = id;
101        }

103        /**
104         * Getter for this Cards Tooltip.
105         * Inherited from ICard.
106         *
107         * @return This Cards Tooltip.
108         */
109        @Override
110        public Tooltip getTooltip ()
111        {
112            return this._tooltip;
113        }
```

```
    /**
     * Setter for this Cards Tooltip.
     *
     * @param newTooltip This Cards Tooltip will be set to the given Tooltip.
     */
    protected void setTooltip (Tooltip newTooltip)
    {
        this._tooltip = newTooltip;
    }

    /**
     * Getter for this Cards Description.
     * Inherited from ICard.
     *
     * @return This Cards Description as a String.
     */
    @Override
    public String getDescription ()
    {
        return this._description;
    }

    /**
     * Setter for this Cards Description.
     *
     * @param newDescription This Cards Description will be set to the given
     *                       String.
     */
    protected void setDescription (String newDescription)
    {
        this._description = newDescription;
    }

    /**
     * Getter for this Cards Name.
     *
     * @return This Cards Name.
     */
    public String getCardName ()
    {
        return this._cardName;
    }

    /**
     * Setter for this Cards Name.
     *
     * @param newCardName This Cards Name will be set to the given String.
     */
    protected void setCardName (String newCardName)
    {
        this._cardName = newCardName;
    }

    /**
     * Getter for this Cards Image.
     * Inherited from ICard.
     *
     * @return This Cards representing Image.
     */
    @Override
    public Image getImage ()
    {
        return _image;
    }
```

```
    /**
     * Setter for this Cards Image.
     * Will set this Cards Image to the Image found on the given Path.
     *
     * If Possible, only use the Paths defined in our Image Interfaces.
     *
     * @param path The Path to the Image File.
     */
    public void setImage (String path)
    {
        try
          {
            this._image = new Image(new FileInputStream(path));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameObject.class.getName())
                                                    .log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Getter for this Cards Mana Cost.Inherited from ICard.A Cards Mana Cost is
     * effected by the current Shift of Flux on the Flux
     * Meter.
     * This is implemented in this Function.
     *
     * @return The Mana Cost of this Card influenced by the current Flux
     *         Situation.
     */
    @Override
    public int getManaCost ()
    {

        //Checks if this Cards Type is Law, Chaos or Neutral.
        switch (this._type)
          {
            //This Cards Type is Law.
            case Law:

                //Checks if the Mana Cost would be reduced below 0.
                if (this._manaCost + FluxMeter.getInstance().getFlux() <= 0)
                  {

                    //Mana Cost cannot be negative, so we return 0.
                    return 0;
                  }

                //returns the Mana Cost as discounted or more expensive,
                //depending on Flux.
                return this._manaCost + FluxMeter.getInstance().getFlux();

            //This Cards Type is Law.
            case Neutral:

                //Mana Cost is not influenced by Flux if the Card Type is Neutral.
                return this._manaCost;

            //This Cards Type is Law.
            case Chaos:

                //Checks if the Mana Cost would be reduced below 0.
                if (this._manaCost - FluxMeter.getInstance().getFlux() <= 0)
                  {

                    //Mana Cost cannot be negative, so we return 0.
```

```
                    return 0;
                  }

                //returns the Mana Cost as discounted or more expensive,
                //depending on Flux.
                return this._manaCost - FluxMeter.getInstance().getFlux();
            }
        return -1;
    }

    /**
     * Checks if the Mana Cost of this Card has been increased or decreased by
     * the current Flux Situation.
     * Inherited from ICard.
     *
     * @return If the Mana Cost has been increased: 1
     *         If the Mana Cost has been decreased: 1
     *         If nothing happened to the Mana Cost: 0
     */
    @Override
    public int getManaCostIncreased ()
    {

        //Checks if the Mana Cost has been increased.
        if (this.getManaCost() > this._manaCost)
          {
            return 1;
          }

        //Checks if the Mana Cost has been increased.
        else if (this.getManaCost() < this._manaCost)
          {
            return -1;
          }

        //Nothing happened to the Mana Cost.
        return 0;
    }

    /**
     * Setter for the Base Mana Cost Value of this Card.
     *
     * @param _manaCost The Base Mana Cost Value of this Card is set to the
     *                  given Value.
     */
    public void setManaCost (int _manaCost)
    {
        this._manaCost = _manaCost;
    }

    /**
     * Getter for this Cards Cast Range.
     *
     * @return This Cards Cast Range.
     */
    public int getCastRange ()
    {
        return this.castRange;
    }

    /**
     * Setter for this Cards Cast Range Value.
     *
     * @param castRange This Cards Cast Range Value is set to the given Value;
     */
    public void setCastRange (int castRange)
```

```
    {
        this.castRange = castRange;
    }

    /**
     * Getter for the Flux Change Parameter of this Card.
     *
     * @return This Cards Flux Change Value.
     */
    public int getFluxChange ()
    {
        return this._fluxChange;
    }

    /**
     * Setter for this Cards Flux Change Value.
     *
     * @param _fluxChange This Cards Flux Change Value is set to the given
     *                    Value.
     */
    public void setFluxChange (int _fluxChange)
    {
        this._fluxChange = _fluxChange;
    }

    /**
     * Getter for this Cards Effect.
     *
     * @return This Cards Effect.
     */
    public ICardEffect getCardEffect ()
    {
        return this._cardEffect;
    }

    /**
     * Setter for this Cards Effect.
     *
     * @param _cardEffect This Cards Effect will be set to the given Effect.
     */
    public void setCardEffect (ICardEffect _cardEffect)
    {
        this._cardEffect = _cardEffect;
    }

    /**
     * Getter for this Cards Type.
     * Inherited from ICard.
     *
     * @return This Cards Type.
     */
    @Override
    public CardType getType ()
    {
        return this._type;
    }

    /**
     * Setter for this Cards Type.
     *
     * @param _type This Cards Type will be set to the given Type.
     */
    public void setType (CardType _type)
    {
        this._type = _type;
    }
```

```
379     /**
380      * Prints out all relevant Information about the card into the Console.
381      */
382     public void printCardInfo ()
383     {
384 //          System.out.println("Card-Name: " + this._cardName);
385 //          System.out.println("Mana-Cost: " + this._manaCost);
386 //          System.out.println("Flux Value: " + Math.abs(this._fluxChange) + "
387 //          towards " + this.getType());
388     }

390     /**
391      * This Function is to be overwritten. It is always executed when the card
392      * is used.
393      */
394     protected void onCast ()
395     {

397     }

399     /**
400      * Executes this Cards Effect as well as applying mana and flux
401      * changes.
402      *
403      * @return True if the Cards Effect could be executed successfully.
404      */
405     @Override
406     public boolean use ()
407     {

409         //Checks if the Effect was executed successfully.
410         if (this._cardEffect.cast())
411           {

413             //Changes Flux Situation depending on this Cards Type.
414             switch (this._type)
415               {
416                 case Law:
417                     FluxMeter.getInstance().addFlux(-this._fluxChange);
418                     break;
419                 case Neutral:
420                     break;
421                 case Chaos:
422                     FluxMeter.getInstance().addFlux(this._fluxChange);
423                     break;
424               }

426             //Executes any onCast Effects of this Card.
427             this.onCast();

429             //Card was used successfully.
430             return true;
431           }
432         else
433           {
434             //Card could not be used.
435             return false;
436           }
437     }

439     /**
440      * Executes this Cards Effect on a target Tile as well as applying mana and
441      * flux changes.
442      *
443      * @param target The target Tile.
```

```
     *
     * @return True if the Cards Effect could be executed successfully.
     */
    @Override
    public boolean use (Tile target)
    {
        //Checks if the target Tile is in this Cards Cast Range.
        if (!MapController.getInstance().getMap().getTilesInRangeExceptObstacles
            (GameController.getInstance().getCurrentTurnController()
                .getCurrentPlayer().getCharacterInstance().getTile(),
                                                this.castRange).contains(target))
          {
            return false;
          }

        //Checks if the Effect was executed successfully.
        if (this._cardEffect.cast(target))
          {

            //Changes Flux Situation depending on this Cards Type.
            switch (this._type)
              {
                case Law:
                    FluxMeter.getInstance().addFlux(-this._fluxChange);
                    break;
                case Neutral:
                    break;
                case Chaos:
                    FluxMeter.getInstance().addFlux(this._fluxChange);
                    break;
              }

            //Executes any onCast Effects of this Card.
            this.onCast();

            //Card was used successfully.
            return true;
          }
        else
          {
            //Card could not be used.
            return false;
          }
    }

    /**
     * Executes this Cards Effect on multiple target Tiles as well as applying
     * mana and flux changes.
     *
     * @param target The target Tiles
     *
     * @return True if the Cards Effect could be executed successfully.
     */
    @Override
    public boolean use (Tile target, int playerID)
    {

        //Checks if the target Tile is in this Cards Cast Range.
        if (!MapController.getInstance().getMap().getTilesInRangeExceptObstacles
                (GameController.getInstance().getCurrentTurnController()
                    .getCurrentPlayer().getCharacterInstance().getTile(),
                                                this.castRange).contains(target))
          {
            return false;
          }
```

```
        //Checks if the Effect was executed successfully.
        if (this._cardEffect.cast(target, playerID))
          {

            //Changes Flux Situation depending on this Cards Type.
            switch (this._type)
              {
                case Law:
                    FluxMeter.getInstance().addFlux(-this._fluxChange);
                    break;
                case Neutral:
                    break;
                case Chaos:
                    FluxMeter.getInstance().addFlux(this._fluxChange);
                    break;
              }

            //Executes any onCast Effects of this Card.
            this.onCast();

            //Card was used successfully.
            return true;
          }
        else
          {
            //Card could not be used.
            return false;
          }
    }

    /**
     * Transforms this Card into a String that can be used to Display relevant
     * Information about this Card.
     *
     * @return This Card as a String.
     */
    @Override
    public String toString ()
    {
        return (this._cardName + " Cost: " + this.getManaCost() + " Effect: "
                                                + this._cardEffect.toString());
    }

}
```

6.9.2 CardEffectAdapter

Listing 35: CardEffectAdapter.java

```
package Model.Card;

import Model.Map.Tile;
import java.util.ArrayList;
import java.util.List;

/**
 * CardEffectAdapter is the Blueprint for all kinds of Card effects. all effects
 * must extend this class. effects can be cast.
 */
public abstract class CardEffectAdapter implements ICardEffect
{
    /**
     * List with all EffectTags of this CardEffect.
     */
    protected List<EffectTag> _tags = null;
```

```
    /**
     * name of this CardEffect.
     */
    protected String _name = null;

    /**
     * CardEffectAdapter is the Blueprint for all kinds of Card effects. all
     * effects must extend this class. effects can be cast. This class is not
     * supposed to be initiated.
     */
    public CardEffectAdapter ()
    {
        this._tags = new ArrayList<>();
    }

    /**
     * Getter for the name of this CardEffect.
     *
     * @return - name of this CardEffect.
     */
    public String getName ()
    {
        return this._name;
    }

    /**
     * Setter for the name of this CardEffect.
     *
     * @param _name - the name of this CardEffect.
     */
    protected void setName (String _name)
    {
        this._name = _name;
    }

    /**
     * the cardEffect is implemented here.
     *
     * @return True if the effect was cast successfully
     */
    @Override
    public boolean cast ()
    {
        return true;
    }

    /**
     * Adds a new Tags to the List of tags of this Cardeffect.
     *
     * @param tag - the new tag
     */
    public void addTag (EffectTag tag)
    {
        if (!this._tags.contains(tag))
          {
            this._tags.add(tag);
          }
    }

    /**
     * This Method proofs if a given tag is attached to the list of this
     * cardeffects tags.
     *
     * @param tag - the tag that you want to proof.
     *
     * @return - true if the tag is attached to this CardEffect.
```

```
     */
    protected boolean hasTag (EffectTag tag)
    {
        return this._tags.contains(tag);
    }

    /**
     * the cardEffect is implemented here for situations where a target Tile is
     * given.
     *
     * @param target the target Tile.
     *
     * @return - true if the cast was succesful
     */
    @Override
    public boolean cast (Tile target)
    {
        return true;
    }

    /**
     * the cardeffect is implemented here for the situation where a target
     * tile and a plers id are given.
     *
     * @param target   - the target tile
     * @param playerID - the players id
     *
     * @return - true if cast was succesful
     */
    @Override
    public boolean cast (Tile target, int playerID)
    {
        return true;
    }

    /**
     * Override of the method toString().
     *
     * @return the name of this effect.
     */
    @Override
    public String toString ()
    {
        return this._name;
    }

}
```

6.9.3 CardStack

Listing 36: CardStack.java

```
package Model.Card;

import Util.Stack;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

/**
 * The CardStack is a stack of cards where you can take off
 * or onto cards from the top or from a random position.
 */
public class CardStack extends Stack
{
    /**
```

```
15      * used to  implement the cardStack.
16      */
17     private List<ICard> _stack = null;

19     /**
20      * default constructor.
21      */
22     public CardStack ()
23     {
24         this._stack = new ArrayList();
25     }

27     /**
28      * constructs a CardStack with a default size.
29      *
30      * @param size the desired default size
31      */
32     public CardStack (int size)
33     {
34         this._stack = new ArrayList<>(size);
35     }

37     /**
38      * constructs a CardStack with a given list of cards.
39      *
40      * @param cards - the list of given cards
41      */
42     public CardStack (List<ICard> cards)
43     {
44         this._stack = cards;
45     }

47     /**
48      * Pops a ICard from a random position in the _stack.
49      *
50      * @return - A random ICard.
51      */
52     public ICard drawRandom ()
53     {
54         if (this._stack.size() < 1)
55           {
56 //            System.out.println("Stack is empty");
57             return null;
58           }
59         Random rand = new Random();
60         int randInt = 0;
61         int length = this._stack.size();
62         if (length > 1)
63           {
64             randInt = rand.nextInt(length - 1);
65           }
66         ICard randICard = this._stack.get(randInt);
67         this._stack.remove(randInt);
68         return randICard;
69     }

71     /**
72      * Pops a ICard from the top of the _stack.
73      *
74      * @return - The top ICard.
75      */
76     public ICard pop ()
77     {
78         int length = this._stack.size();
79         ICard top = this._stack.get(length - 1);
80         this._stack.remove(length - 1);
```

```
        return top;
    }

    /**
     * Puts a new ICard onto the top of the _stack.
     *
     * @param o - The new ICard.
     */
    public void put (ICard o)
    {
        this._stack.add(o);
    }

    /**
     * use this to find out how many Cards are in this _stack.
     *
     * @return - amount of cards that are in this stack.
     */
    public int getSize ()
    {
        int size = 0;
        for (ICard c : this._stack)
          {
            if (c != null)
              {
                size++;
              }
          }
        return size;
    }

    /**
     * getter for all cards of this stack in a ICard array
     *
     * @return - ICard array with all cards of this stack.
     */
    public ICard[] getCards ()
    {
        ICard[] allCards = new ICard[this._stack.size()];
        int i = 0;
        for (ICard c : this._stack)
          {
            allCards[i] = c;
            i++;
          }
        return allCards;
    }

    /**
     * returns a card at a location without removing it from the _stack.
     *
     * @param index - the location of the card
     *
     * @return - the instance of the card
     */
    public ICard find (int index)
    {
        return this._stack.get(index);
    }

    /**
     * Override of the method toString().
     *
     * @return - this class as a string.
     */
    @Override
```

```
    public String toString ()
    {
        String target = "";
        for (ICard c : this._stack)
          {
            target += c.toString() + " at position: "
                                                + this._stack.lastIndexOf(c) + "\n";
          }
        return target;
    }
}
```

6.9.4 CardType

Listing 37: CardType.java

```
package Model.Card;

/**
 * Enumeration for the different kind of Cards.
 */
public enum CardType
{
    /**
     * Type Law (Internal: 0)
     */
    Law,

    /**
     * Type Law (Internal: 1)
     */
    Neutral,

    /**
     * Type Law (Internal: 2)
     */
    Chaos;

}
```

6.9.5 Deck

Listing 38: Deck.java

```
package Model.Card;

/**
 * This class implements a Deck of Cards used for the player to cut cards from it.
 */
public class Deck
{
    /**
     * the deck implemented with a cardstack.
     */
    private CardStack _deck = null;

    /**
     * Default Constructor.
     */
    public Deck ()
    {
        this._deck = new CardStack();
    }

    /**
```

```
     * constructs a new deck with the type of the given wand.
     *
     * @param wand - the wand that the cards should be the type of
     */
    public Deck (Wand wand)
    {
        this._deck = new CardStack(wand.getCards());
    }

    /**
     * use this to shuffle or add a ICard to a Deck.
     *
     * @param c - This ICard will be put on top of the Deck.
     */
    public void addCard (ICard c)
    {
        this._deck.put(c);
    }

    /**
     * determines the amount of cards left in the deck.
     *
     * @return - the amount of remaining cards
     */
    public int getSize ()
    {
        return this._deck.getSize();
    }

    /**
     * Removes the top card of the deck and returns it.
     *
     * @return - the top card of the deck.
     */
    public ICard drawTopCard ()
    {
        ICard top = this._deck.pop();
        return top;
    }

    /**
     * Removes a random Card from anywhere in the deck and returns it
     *
     * @return - A random Card.
     */
    public ICard drawRandomCard ()
    {
        ICard rand = this._deck.drawRandom();
        if (rand == null)
          {
//            System.out.println("The Deck is empty");
            return null;
          }
        return rand;
    }

    /**
     * Override of the method toString().
     *
     * @return - the class as a string.
     */
    @Override
    public String toString ()
    {
        String target = "Deck: \n";
        int i = 0;
```

```
        for (ICard c : this._deck.getCards())
          {
            target += c.toString() + ", Position: " + i + "\n";
            i++;
          }
        return target;
    }

}
```

6.9.6 EffectTag

Listing 39: EffectTag.java

```
package Model.Card;

/**
 * Enumeration of all Tags that can be added to an Effect.
 */
public enum EffectTag
{
    /**
     * An Effect with this tag cannot damage the Wizard.
     */
    WIZARD_IMMUNE
}
```

6.9.7 GenericCardEffect

Listing 40: GenericCardEffect.java

```
package Model.Card;

import static Resource.Constants.CardConstants.CardConstants.DEFAULT_CARD_EFFECT_NAME;
import java.util.ArrayList;

/**
 * The Generic Card Effect defines a CardEffect whose Effect should be
 * overwritten upon creation. We do this, to not have to create a new class for
 * every Card Effect.
 */
public class GenericCardEffect extends CardEffectAdapter
{
    /**
     * The Generic Card Effect defines a CardEffect whose Effect should be
     * overwritten upon creation.
     */
    public GenericCardEffect ()
    {
        this._tags = new ArrayList<EffectTag>();
        this._name = DEFAULT_CARD_EFFECT_NAME;
    }
}
```

6.9.8 Hand

Listing 41: Hand.java

```
package Model.Card;

import Control.Controllers.GameController;
import Control.Network.Packets.Packet;
import Model.Map.Tile;
```

```
import Model.Player;
import static Resource.Constants.GeneralConstants.ControllerConstants.SPACE;
import static Resource.Constants.GeneralConstants.NetworkConstants.*;
import Resource.Constants.GeneralConstants.PlayerConstants;
import static Resource.Constants.GeneralConstants.PlayerConstants.PLAYER_DEFAULT_STARTING_HAND_SIZE;
import Resource.Images.Images;
import View.UIElements.CardView;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.image
                                                .Image;

/**
 * The Hand contains all cards that are playable for the Player in this match.
 */
public class Hand
{

    /**
     * The Card deck where the player takes cards off.
     */
    private Deck _deck = null;

    /**
     * All the _cards currently in the hand of the _player (no more than the
     * maximum hand size).
     */
    private List<ICard> _cards = null;

    /**
     * the current size of the cards that are on the players hand.
     */
    private int _currentSize = 0;

    /**
     * The Player this Hand is related to.
     */
    private Player _player = null;

    /**
     * The CardView of this Hand.
     */
    private List<CardView> _cardViews = null;

    /**
     * Constructor that constructs a new hand with given player and deck.
     *
     * @param player this Hands owner _player.
     * @param deck   the Deck this Hand will draw from.
     */
    public Hand (Player player, Deck deck)
    {
        this._deck = deck;
        this._cards = new ArrayList<ICard>();
        this._player = player;
        this._cardViews = new ArrayList<>();
        this.drawCard(PLAYER_DEFAULT_STARTING_HAND_SIZE);
    }

    /**
     * Removes a card from the top of the players _deck, and adds it to his
```

```
     * hand, if there is enough space.
     */
    public void drawCard ()
    {
        if (_currentSize == PlayerConstants.PLAYER_MAXIMUM_HAND_SIZE)
          {
            return;
          }

        ICard drawnCard = this._deck.drawRandomCard();

        if (drawnCard == null)
          {
            return;
          }

        this._cards.add(drawnCard);
        this._cardViews.add(new CardView(drawnCard.getImage(), drawnCard, false));

        if (GameController.getInstance().getCurrentGame().isMultiplayer())
          {
            GameController.getInstance().getClient().sendPackage(new
                                                Packet(HAND + drawnCard.getID()));
          }

        _currentSize++;

    }

    /**
     * Removes a given amount of cards from the top of the players _deck, and
     * adds it to his hand, if there is enough space.
     *
     * @param c - amount of cards that should be drawn
     */
    public void drawCard (int c)
    {
        for (int i = 0; i < c; i++)
          {
            this.drawCard();
          }
    }

    /**
     * Adds a given card to the hand of the player.
     *
     * @param c - the card that should be added
     */
    public void addCard (ICard c)
    {
        this._cards.add(c);
        try
          {
            this._cardViews.add(new CardView(new Image(new FileInputStream(
                                                Images.NO_IMAGE)), c, true));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(Hand.class.getName()).log(Level.SEVERE, null, ex);
          }

        _currentSize++;
    }

    /**
     * Getter for the current size of the hand.
```

```
     *
     * @return - size of the hand
     */
    public int getSize ()
    {
        return this._currentSize;
    }

    /**
     * Use this to find out what card is at what position in the players hand.
     *
     * @param index - starts at 0.
     *
     * @return - returns the Card at the given position.
     */
    public ICard getCard (int index)
    {
        return _cards.get(index);
    }

    /**
     * Getter for the current Cards that the hand contains.
     *
     * @return - current hand cards
     */
    public List<ICard> getCards ()
    {
        return _cards;
    }

    /**
     * Getter for the index of a given Card in the hand.
     *
     * @param c - the card
     *
     * @return - the index of the given card
     */
    public int getIndexOf (ICard c)
    {
        for (int i = 0; i < this._cards.size(); i++)
          {
            if (c == this._cards.get(i))
              {
                return i;
              }
          }
        return -1;
    }

    /**
     * use this to execute a card from this Hand at a given index and with a
     * target Tile.
     *
     * @param index  the given index
     * @param target the target Tile
     *
     * @return True if the card has been played successfully.
     */
    public boolean playCard (int index, Tile target)
    {
        if (this._cards.get(index).use(target))
          {
            this._cards.remove(index);
            this._cardViews.remove(index);
            this._currentSize--;
            return true;
```

```
            }
        return false;
    }

    /**
     * use this to execute a card from this Hand at a given index.
     *
     * @param index - the given index.
     *
     * @return - true if the card was played succesfully.
     */
    public boolean playCard (int index)
    {
        if (this._cards.remove(index).use())
          {
            this._cardViews.remove(index);
            this._currentSize--;
            return true;
          }
        return false;
    }

    /**
     * Removes a card at the given ndex from the hand.
     *
     * @param index - the position of the card that should be removed
     *
     * @return - true if removing was succesful
     */
    public boolean removeCard (int index)
    {
        if (this._cards.get(index) != null)
          {
            this._cards.remove(index);
            this._cardViews.remove(index);
            this._currentSize--;
            return true;
          }
        return false;
    }

    /**
     * Discards a card at a given index.
     *
     * @param index - the index of the card
     */
    public void discard (int index)
    {
        if (GameController.getInstance().getCurrentGame().isMultiplayer())
          {
            GameController.getInstance().getClient().sendPackage(new Packet(
                     HAND_MINUS_ONE + index + SPACE + this._player.getOwnerNumber()));
          }
    }

    /**
     * Discards random cards from the hand.
     *
     * @param count - amount of random cards should be discarded
     */
    public void discardRandomCards (int count)
    {
        Random r = new Random();
        for (int i = 0; i < count; i++)
          {
            int rand = r.nextInt(this._cards.size() - 1);
```

```
                this.discard(rand);
            }
    }

    /**
     * Getter for the CardView.
     *
     * @return - CardView
     */
    public List<CardView> getCardViews ()
    {
        for (CardView cv : this._cardViews)
          {
            cv.refresh();
          }
        return this._cardViews;
    }

    /**
     * getter for the deck thois hand takes cards off.
     *
     * @return - the deck related to this hand.
     */
    public Deck getDeck ()
    {
        return this._deck;
    }

    /**
     * Override of the method toString().
     *
     * @return - all important information about this class in one string
     */
    @Override
    public String toString ()
    {
        String output = EMPTY;

        for (int i = 0; i < this._currentSize; i++)
          {
            output += this._cards.get(i).toString() + " at position " + i + "\n";
          }
        return "Hand of Player" + GameController.getInstance()
            .getCurrentTurnController().getCurrentPlayer().getOwnerNumber()
                + "\nContains " + this._currentSize + " cards: \n" + output
                    + "Your mana: " + GameController.getInstance()
                        .getCurrentTurnController().getCurrentPlayer().getMana();
    }

}
```

6.9.9 ICard

Listing 42: ICard.java

```
package Model.Card;

import Model.Map.Tile;
import View.Tooltip.Tooltip;
import javafx.scene.image.Image;

/**
 * This interface defines some method heads for cards.
 */
public interface ICard
{
```

```
13      /**
14       * Override this to implement the card.
15       *
16       * @return - true if card was succesful used
17       */
18      public boolean use ();

20      /**
21       * Override this to implement the card with a given target Tile.
22       *
23       * @param target - the target Tile
24       *
25       * @return - true if card was succesful used
26       */
27      public boolean use (Tile target);

29      /**
30       * Override this to implement the card with a given target tile and playerID
31       *
32       * @param target   - the target tile
33       * @param playerID - the players id
34       *
35       * @return - true if card was succesful used
36       */
37      public boolean use (Tile target, int playerID);

39      /**
40       * Getter for the id of this card.
41       *
42       * @return - the id
43       */
44      public int getID ();

46      /**
47       * Getter for the ManaCost of the implementing Card.
48       *
49       * @return - the mana cost of the implementing card
50       *
51       */
52      public int getManaCost ();

54      /**
55       * Getter for the image of the implementing card.
56       *
57       * @return - cards image
58       */
59      public Image getImage ();

61      /**
62       * Getter for the type of the implementing card.
63       *
64       * @return - type of this card
65       */
66      public CardType getType ();

68      /**
69       * Getter for the increased Manacost by this card.
70       *
71       * @return - the amount how much the manacost increases by this card
72       */
73      public int getManaCostIncreased ();

75      /**
76       * Getter for the Tooltip of the implementing.
77       *
```

```
     * @return - the cards tooltip
     */
    public Tooltip getTooltip ();

    /**
     * Getter for the Description of the implementing Card.
     *
     * @return - cards description
     */
    public String getDescription ();

}
```

6.9.10 ICardEffect

Listing 43: ICardEffect.java

```
package Model.Card;

import Model.Map.Tile;

/**
 * Unites All Card Effects into one interface. All Card Effects must inherit
 * this interface.
 */
public interface ICardEffect
{
    /**
     * Override this to implement the possibilitie of adding a tag to the
     * cardeffect.
     *
     * @param tag - the tag that should be added to the effect
     */
    public void addTag (EffectTag tag);

    /**
     * Override this to implement this effect.
     *
     * @return - true if casting this cardeffect was succesful
     */
    public boolean cast ();

    /**
     * Override this to implement this effect with a given target Tile.
     *
     * @param target - the target Tile
     *
     * @return - true if casting this cardeffect was succesful
     */
    public boolean cast (Tile target);

    /**
     * Override this to implement this effect with a given target tile and
     * playerID
     *
     * @param target   - the target tile
     * @param playerID - the playerID
     *
     * @return - true if casting this card effect was succesfull
     */
    public boolean cast (Tile target, int playerID);

}
```

6.9.11 Wand

Listing 44: Wand.java

```
package Model.Card;

import Model.Card.AllCards.CardFactory;
import static Resource.Constants.CardConstants.CardConstants.DEFAULT_WAND_NAME;
import View.Tooltip.Tooltip;
import View.Tooltip.WandTooltip;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.image.Image;

/**
 * A Wand is a List of Cards of the same Type.
 */
public abstract class Wand
{
    /**
     * List with all cards that are in this wand.
     */
    private List<ICard> _cards = null;

    /**
     * the image of the Wand.
     */
    private Image _image = null;

    /**
     * name of the wand.
     */
    private String _name = null;

    /**
     * The Tooltip of this wand, displayed to provide information about it.
     */
    private Tooltip _tooltip = null;

    /**
     * Default Constructor.
     */
    public Wand ()
    {
        this._cards = new ArrayList<ICard>();
        this._name = DEFAULT_WAND_NAME;
        this._tooltip = new WandTooltip(DEFAULT_WAND_NAME);
    }

    /**
     * Constructor that sets the name of this wand to the given string.
     *
     * @param name - the desired name for this wand
     */
    public Wand (String name)
    {
        this._cards = new ArrayList<ICard>();
        this._name = name;
        this._tooltip = new WandTooltip(DEFAULT_WAND_NAME);
        this._tooltip.setTitle(name);
    }

    /**
     * Setter for the Tooltip of this Wand.
     *
```

```
     * @param t - the tooltip
     */
    protected void setTooltip (Tooltip t)
    {
        this._tooltip = t;
    }

    /**
     * Getter for the Wands tooltip.
     *
     * @return - the wands tooltip
     */
    protected Tooltip getTooltip ()
    {
        return this._tooltip;
    }

    /**
     * Setter for the Wands image.
     *
     * @param ims - Path to the wands image.
     */
    protected void setImage (String ims)
    {
        try
          {
            this._image = new Image(new FileInputStream(ims));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(Wand.class.getName()).log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Getter for the Wands image.
     *
     * @return - the wands image
     */
    public Image getImage ()
    {
        return this._image;
    }

    /**
     * Use this to add cards to a Wand, cards added to a wand can be drawn by
     * the player.
     *
     * @param id    - the public id of the card to be added
     * @param count - the amount of instances of the given card to be added to
     *               this wand.
     */
    public void addCards (int id, int count)
    {
        for (int i = 0; i < count; i++)
          {
            this._cards.add(CardFactory.getNewInstance(id));
          }
        this._tooltip.addInfo(CardFactory.getNewInstance(id).getClass()
                                .getSimpleName(), ((Integer) count).toString());
    }

    /**
     * Getter for the name of the wand.
     *
     * @return - name of this wand
```

```
     */
    public String getName ()
    {
        return this._name;
    }

    /**
     * Getter for the Cards that the Wand contains.
     *
     * @return - all cards of this wand.
     */
    public List<ICard> getCards ()
    {
        return this._cards;
    }

    /**
     * Setter for the Cards that the Wand should contain.
     *
     * @param cards - List of ICards
     */
    protected void setCards (List<ICard> cards)
    {
        this._cards = cards;
    }

    /**
     * Displayes the Tooltip of this Wand.
     */
    public void showTooltip ()
    {
        this._tooltip.show();
    }

    /**
     * Hides the Tooltip of this wand.
     */
    public void hideTooltip ()
    {
        this._tooltip.hide();
    }

    /**
     * initializes the wand, has to be overwritten in every subclass.
     */
    public abstract void init ();

}
```

6.9.12 WandFactory

Listing 45: WandFactory.java

```
package Model.Card;

import Model.Card.Wands.AirWand;
import Model.Card.Wands.CreatureWand;
import Model.Card.Wands.RandomWand;
import Model.Card.Wands.UndeadWand;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * The Wandfactory provides all Wands in one class
 * to choose one or get a random wand.
```

```
 */
public class WandFactory
{

    /**
     * This classArray stores all wand classes.
     */
    private static Class[] _wands =
      {
        AirWand.class,
        CreatureWand.class,
        UndeadWand.class,
        //RandomWand HAS to be the last class in this list. Do NOT reorganize! ; ok
        RandomWand.class
      };

    /**
     * Getter for the wand with the given index.
     *
     * @param id - index to choose the right wand
     * @return - the wand of the given index
     */
    public static Wand getWand (int id)
    {
        try
          {
            Wand w = (Wand) _wands[id].newInstance();
            w.init();
            return w;
          }
        catch (InstantiationException | IllegalAccessException ex)
          {
            Logger.getLogger(WandFactory.class.getName())
                                                .log(Level.SEVERE, null, ex);
          }
        return null;
    }

    /**
     * Getter for a random Wand.
     *
     * @return - a random Wand
     */
    public static Wand getRandomWand ()
    {
        Random r = new Random();

        //Excludes RandomWand with length - 1.
        return getWand(r.nextInt(_wands.length - 1));
    }

    /**
     * Getter for the amount of wands
     *
     * @return - amount of all wands
     */
    public static int getWandCount ()
    {
        return _wands.length;
    }

}
```

6.10 Model.Card.AllCards

6.10.1 CardFactory

Listing 46: CardFactory.java

```
package Model.Card.AllCards;

import Model.Card.AllCards.Spells.Chaos.Vengeance;
import Model.Card.AllCards.Spells.Law.Decree;
import Model.Card.AllCards.Spells.Law.DivineBow;
import Model.Card.AllCards.Spells.Law.DivineShield;
import Model.Card.AllCards.Spells.Law.DivineSword;
import Model.Card.AllCards.Spells.Neutral.MagicBolt;
import Model.Card.AllCards.Spells.Neutral.WindWalker;
import Model.Card.AllCards.Summons.Chaos.*;
import Model.Card.AllCards.Summons.Law.IcarusTower;
import Model.Card.AllCards.Summons.Law.Paladin;
import Model.Card.AllCards.Summons.Law.Pegasus;
import Model.Card.AllCards.Summons.Law.SapphireDragon;
import Model.Card.AllCards.Summons.Neutral.AirElemental;
import Model.Card.AllCards.Summons.Neutral.Eagle;
import Model.Card.AllCards.Summons.Neutral.Giant;
import Model.Card.AllCards.Summons.Neutral.Lion;
import Model.Card.ICard;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * The CardFactory produces Cards of the given index.
 */
public class CardFactory
{

    /**
     * ClassArray with all Cards in it, used for initializing the Wands,
     * and decoding packets.
     */
    private static final Class[] CARDS =
      {
        // All Neutral Minions:
        Lion.class, //0
        Eagle.class, //1
        AirElemental.class, //2
        Giant.class, //3
        // All Law Minions:
        SapphireDragon.class, //4
        IcarusTower.class, //5 ;
        Pegasus.class, //6 ;
        Paladin.class, //7 ;
        // All Chaos Minions:
        Manticore.class, //8
        Vampire.class, //9 ;
        Skeleton.class, //10 ;
        Hellhound.class, //11 ;
        Hydra.class, //12 :
        Zombie.class, //13 ;
        // All Neutral Spells:
        MagicBolt.class, //14 ;
        WindWalker.class, //15 ;
        // All Law Spells:
        Decree.class, //16 ;
        DivineBow.class, //17 ;
        DivineShield.class, //18 ;
        DivineSword.class, //19 ;
```

```
        // All Chaos Spells:
        Vengeance.class //20 ;
      };

    /**
     * Returns a new Card instance of the class with the given index.
     *
     * @param id - the index of the card you want to produce.
     *
     * @return - the card of the given index
     */
    public static ICard getNewInstance (int id)
    {
        try
          {
            return (ICard) CARDS[id].newInstance();
          }
        catch (InstantiationException | IllegalAccessException ex)
          {
            Logger.getLogger(CardFactory.class.getName())
                                                .log(Level.SEVERE, null, ex);
          }
        return null;
    }

}
```

6.11 Model.Card.AllCards.Spells

6.11.1 AttackSpell

Listing 47: AttackSpell.java

```
package Model.Card.AllCards.Spells;

import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Model.Card.CardEffectAdapter;
import Model.GameObject.Attribute;
import Model.GameObject.CreatureAdapter;
import Model.GameObject.GameObject;
import Model.GameObject.IAttackable;
import Model.Map.Tile;
import Resource.Images.Animations;

/**
 * An AttackSpell is a Spell that attacks a hostile minion when it is casted,
 * it can only be casted on hostile minions..
 */
public class AttackSpell extends Spell
{

    /**
     * The tile where this AttackSpell should attack.
     */
    private Tile _targetTile = null;

    /**
     * Default Constructor.
     */
    public AttackSpell ()
    {
        this.setCardEffect(new AttackSpellEffect());
    }
```

```
33      /**
34       * Getter for the target of this AttackSpell.
35       *
36       * @return this spells target tile.
37       */
38      public Tile getTarget ()
39      {
40          return this._targetTile;
41      }

43      /**
44       * This is executed when the spell is cast.
45       */
46      @Override
47      protected void onCast ()
48      {
49          super.onCast();
50          this._targetTile = ((AttackSpellEffect) this.getCardEffect()).getTarget();
51      }

53      /**
54       * The effect of AttackSpells is always the same.
55       */
56      protected class AttackSpellEffect extends CardEffectAdapter
57      {

59          /**
60           * The attack value of this AttackSpellEffect to make damage on the
61           * hostile minion.
62           */
63          private int _attackValue = 0;

65          /**
66           *
67           */
68          private Tile _targetTile = null;

70          /**
71           * Default Constructor.
72           */
73          public AttackSpellEffect ()
74          {
75          }

77          /**
78           * This is the Amount of damage this spell deals.
79           *
80           * @param v the desired amount of damage.
81           */
82          public void setAttackValue (int v)
83          {
84              this._attackValue = v;
85          }

87          /**
88           * This is this spells effect name.
89           *
90           * @param name the desired name
91           */
92          public void setSpellName (String name)
93          {
94              this.setName(name);
95          }

97          /**
98           * Getter for the Target Tile of this AttackSpellEffect.
```

```
         *
         * @return - this effects target tile
         */
        public Tile getTarget ()
        {
            return this._targetTile;
        }

        /**
         * Getter for the AttackSpellEffect itself as a CardEffect.
         *
         * @return - returns this spell effect as a card effect.
         */
        public AttackSpellEffect get ()
        {
            return this;
        }

        /**
         * The Effect of this Card Effect is defined here.
         *
         * @param target - The target tile
         *
         * @return - true if the effect was executed successfully.
         */
        @Override
        public boolean cast (Tile target)
        {
            if (CreatureAdapter.class.isInstance(target.getGameObject()))
              {
                if (GameController.getInstance().getCurrentTurnController()
                    .getCurrentPlayer().getOwnerNumber() == ((CreatureAdapter)
                                                target.getGameObject()).getOwner())
                  {
                    return false;
                  }

                if (((CreatureAdapter) target.getGameObject())
                                                .hasAttribute(Attribute.Structure))
                  {
                    return false;
                  }

                if (IAttackable.class.isInstance(target.getGameObject()))
                  {
                    IAttackable victim = (IAttackable) target.getGameObject();

                    AnimationController.getInstance().playAnimationOnTile(
                          Animations.SPELLHIT, ((GameObject) victim).getTile());
                    this._targetTile = target;
                    return true;
                  }
              }
            return false;
        }

        /**
         * The Effect of this Card Effect is defined here.
         *
         * @param target   The tile this effect is cast on.
         * @param playerID The PlayerID of the Player that cast the effect.
         *
         * @return True if the Effect was cast successfully.
         */
        @Override
        public boolean cast (Tile target, int playerID)
```

```
        {
            if (CreatureAdapter.class.isInstance(target.getGameObject()))
              {
                if (playerID == ((CreatureAdapter) target.getGameObject())
                                                                .getOwner())
                  {
                    return false;
                  }

                if (((CreatureAdapter) target.getGameObject()).hasAttribute(
                                                        Attribute.Structure))
                  {
                    return false;
                  }

                if (IAttackable.class.isInstance(target.getGameObject()))
                  {
                    IAttackable victim = (IAttackable) target.getGameObject();

                    AnimationController.getInstance().playAnimationOnTile(
                          Animations.SPELLHIT, ((GameObject) victim).getTile());

                    this._targetTile = target;
                    return true;

                  }
                this._targetTile = target;
                return true;
              }
            return false;
        }

    }

}
```

6.11.2 Spell

Listing 48: Spell.java

```
package Model.Card.AllCards.Spells;

import Model.Card.CardAdapter;
import View.Tooltip.CardTooltip;

/**
 * A Spell is a  Effect that can harm,
 * heal or upgrade a minion where it is casted on.
 */
public class Spell extends CardAdapter
{
    /**
     * displays if the spell needs to have line of sight from the wizard to the
     * tile where the spell should be casted.
     */
    private boolean _requiresLineOfSight = false;

    /**
     * Invoke this if you want to know if this spell requires line of sight or not.
     *
     * @return - true if this spell requires line of sight to hit its target
     */
    public boolean requiresLineOfSight ()
    {
        return this._requiresLineOfSight;
    }
```

```
    /**
     * Setter to enable or disable this spell to require line of sight.
     *
     * Default: false
     *
     * @param b - set this to true if the spell should only work with line of
     *           sight to the target
     */
    protected void setRequiresLineOfSight (boolean b)
    {
        this._requiresLineOfSight = b;
    }

    /**
     * Invoke this to set the tooltip of this spell.
     */
    protected void setSpellTooltip ()
    {
        this.setTooltip(new CardTooltip(this.getCardName(), this));
    }
}
```

6.12 Model.Card.AllCards.Spells.Chaos

6.12.1 Vengeance

Listing 49: Vengeance.java

```
package Model.Card.AllCards.Spells.Chaos;

import Control.Controllers.GameController;
import Model.Card.AllCards.Spells.AttackSpell;
import Model.Card.CardType;
import Model.Card.EffectTag;
import Model.GameObject.PlayerCharacter;
import static Resource.Constants.CardConstants.SpellConstants.Chaos.VengeanceSpellConstants.*;
import Resource
                                                        .Images.Images;

/**
 * Vengeance is a very rare but powerful Card with a Attackvalue of 80 and
 * Fluxchange of two.
 */
public class Vengeance extends AttackSpell
{

    /**
     * Default Constructor.
     */
    public Vengeance ()
    {
        this.setType(CardType.Chaos);
        this.setCardName(VENGEANCE_SPELL_CARD_NAME);
        this.setCastRange(VENGEANCE_SPELL_CAST_RANGE);
        this.setManaCost(VENGEANCE_SPELL_MANA_COST);
        this.setFluxChange(VENGEANCE_SPELL_FLUX_CHANGE);
        this.setID(VENGEANCE_SPELL_CARD_ID);
        this.setImage(Images.VENGEANCE);
        this.setDescription(VENGEANCE_SPELL_CARD_DESCRIPTION);
        ((AttackSpell.AttackSpellEffect) this.getCardEffect())
                        .setAttackValue(VENGEANCE_SPELL_CARD_EFFECT_ATTACK_VALUE);
        ((AttackSpell.AttackSpellEffect) this.getCardEffect())
                            .setSpellName(VENGEANCE_SPELL_CARD_EFFECT_SPELL_NAME);
        this.getCardEffect().addTag(EffectTag.WIZARD_IMMUNE);
```

```
        this.setSpellTooltip();
    }

    /**
     * Will be executed when this spell is cast.
     */
    @Override
    protected void onCast ()
    {
        super.onCast();
        if (this.getTarget().getGameObject() == null)
          {
            return;
          }
        if (PlayerCharacter.class.isInstance(this.getTarget().getGameObject()))
          {
            GameController.getInstance().getCurrentTurnController()
                                .getEnemyPlayer().getHand().discardRandomCards(1);
            if (!GameController.getInstance().getCurrentGame().isMultiplayer())
              {
                GameController.getInstance().getCurrentTurnController()
                                .getEnemyPlayer().getHand().discardRandomCards(1);
              }
          }
    }

}
```

6.13 Model.Card.AllCards.Spells.Law

6.13.1 Decree

Listing 50: Decree.java

```
package Model.Card.AllCards.Spells.Law;

import Control.Controllers.GameController;
import Model.Card.AllCards.Spells.AttackSpell;
import Model.Card.CardType;
import Model.Card.EffectTag;
import Model.GameObject.PlayerCharacter;
import static Resource.Constants.CardConstants.SpellConstants.Law.DecreeSpellConstants.*;
import Resource
                                                        .Images.Images;

/**
 * Decree is a very powerful Spell with a Attackvalue of 80 and a Fluxchange of
 * 2, discards a random card from enemys hand.
 */
public class Decree extends AttackSpell
{

    /**
     * Default Constructor.
     */
    public Decree ()
    {
        this.setType(CardType.Law);
        this.setCardName(DECREE_SPELL_CARD_NAME);
        this.setCastRange(DECREE_SPELL_CAST_RANGE);
        this.setManaCost(DECREE_SPELL_MANA_COST);
        this.setID(DECREE_SPELL_CARD_ID);
        this.setFluxChange(DECREE_SPELL_FLUX_CHANGE);
        this.setImage(Images.DECREE);
        this.setDescription(DECREE_SPELL_CARD_DESCRIPTION);
```

```
        ((AttackSpellEffect) this.getCardEffect())
                        .setAttackValue(DECREE_SPELL_CARD_EFFECT_ATTACK_VALUE);
        ((AttackSpellEffect) this.getCardEffect())
                            .setSpellName(DECREE_SPELL_CARD_EFFECT_SPELL_NAME);
        this.getCardEffect().addTag(EffectTag.WIZARD_IMMUNE);
        this.setSpellTooltip();
    }

    /**
     * Will be executed when this spell is cast.
     */
    @Override
    protected void onCast ()
    {
        super.onCast();
        if (this.getTarget().getGameObject() == null)
          {
            return;
          }
        if (PlayerCharacter.class.isInstance(this.getTarget().getGameObject()))
          {
            GameController.getInstance().getCurrentTurnController()
                .getEnemyPlayer().getHand()
                    .discardRandomCards(DECREE_SPELL_ON_CAST_AMOUNT_OF_DISCARD);
            if (!GameController.getInstance().getCurrentGame().isMultiplayer())
              {
                GameController.getInstance().getCurrentTurnController()
                    .getEnemyPlayer().getHand()
                    .discardRandomCards(
                     DECREE_SPELL_ON_CAST_AMOUNT_OF_DISCARD_IS_NOT_MULTIPLAYER);
              }
          }
    }
}
```

6.13.2 DivineBow

Listing 51: DivineBow.java

```
package Model.Card.AllCards.Spells.Law;

import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Control.Controllers.PlayerController;
import Model.Card.AllCards.Spells.Spell;
import Model.Card.CardType;
import Model.Card.GenericCardEffect;
import Model.Map.Tile;
import static Resource.Constants.CardConstants.SpellConstants.Law
                                                        .DivineBowSpellConstants.*;
import Resource.Images.Animations;
import Resource.Images.Images;

/**
 * When Divine Bow is cast on a friendly Wizard it grants +20 to Rangedattack
 * and +3 to Attackrange.
 */
public class DivineBow extends Spell
{

    /**
     * Default Constructor.
     */
    public DivineBow ()
    {
        this.setType(CardType.Law);
```

```
        this.setCardName(DIVINE_BOW_SPELL_CARD_NAME);
        this.setManaCost(DIVINE_BOW_SPELL_MANA_COST);
        this.setFluxChange(DIVINE_BOW_SPELL_FLUX_CHANGE);
        this.setID(DIVINE_BOW_SPELL_CARD_ID);
        this.setCastRange(DIVINE_BOW_SPELL_CAST_RANGE);
        this.setDescription(DIVINE_BOW_SPELL_CARD_DESCRIPTION);
        this.setImage(Images.DIVINE_BOW);
        this.setCardEffect(new GenericCardEffect()
        {
            @Override
            public boolean cast (Tile target)
            {
                GameController.getInstance().getCurrentTurnController()
                    .getCurrentPlayer().getCharacterInstance()
                    .setAttackRange(DIVINE_BOW_SPELL_ON_CAST_SET_ATTACK_RANGE);
                GameController.getInstance().getCurrentTurnController()
                    .getCurrentPlayer().getCharacterInstance()
                    .addRangedAttack(DIVINE_BOW_SPELL_ON_CAST_ADD_RANGED_ATTACK);
                AnimationController.getInstance()
                                    .playAnimationOnTile(Animations.BUFF, target);
                return true;
            }

            @Override
            public boolean cast (Tile target, int playerID)
            {
                PlayerController.getInstance().getPlayer(playerID)
                    .getCharacterInstance()
                    .setAttackRange(DIVINE_BOW_SPELL_ON_CAST_SET_ATTACK_RANGE);
                PlayerController.getInstance().getPlayer(playerID)
                    .getCharacterInstance()
                    .addRangedAttack(DIVINE_BOW_SPELL_ON_CAST_ADD_RANGED_ATTACK);
                AnimationController.getInstance()
                                    .playAnimationOnTile(Animations.BUFF, target);
                return true;
            }

        }
        );
        this.setSpellTooltip();
    }
}
```

6.13.3 DivineShield

Listing 52: DivineShield.java

```
package Model.Card.AllCards.Spells.Law;

import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Control.Controllers.PlayerController;
import Model.Card.AllCards.Spells.Spell;
import Model.Card.CardType;
import Model.Card.GenericCardEffect;
import Model.Map.Tile;
import static Resource.Constants.CardConstants.SpellConstants
                                        .Law.DivineShieldSpellConstants.*;
import Resource.Images.Animations;
import Resource.Images.Images;

/**
 * Divine Shield is a helpful Spell that grants a friendly Wizard +20 physical
 * resistance.
 */
public class DivineShield extends Spell
```

```
{

    /**
     * Default Constructor.
     */
    public DivineShield ()
    {
        this.setType(CardType.Law);
        this.setCardName(DIVINE_SHIELD_SPELL_CARD_NAME);
        this.setManaCost(DIVINE_SHIELD_SPELL_MANA_COST);
        this.setFluxChange(DIVINE_SHIELD_SPELL_FLUX_CHANGE);
        this.setCastRange(DIVINE_SHIELD_SPELL_CAST_RANGE);
        this.setID(DIVINE_SHIELD_SPELL_CARD_ID);
        this.setImage(Images.DIVINE_SHIELD);
        this.setDescription(DIVINE_SHIELD_SPELL_CARD_DESCRIPTION);
        this.setCardEffect(new GenericCardEffect()
        {
            @Override
            public boolean cast (Tile target)
            {
                GameController.getInstance().getCurrentTurnController()
                    .getCurrentPlayer().getCharacterInstance()
                          .addPhysRes(DIVINE_SHIELD_SPELL_ON_CAST_ADD_PHYS_RES);
                AnimationController.getInstance()
                                    .playAnimationOnTile(Animations.BUFF, target);
                return true;
            }

            @Override
            public boolean cast (Tile target, int playerID)
            {
                PlayerController.getInstance().getPlayer(playerID)
                    .getCharacterInstance()
                        .addPhysRes(DIVINE_SHIELD_SPELL_ON_CAST_ADD_PHYS_RES);
                AnimationController.getInstance()
                                    .playAnimationOnTile(Animations.BUFF, target);
                return true;
            }
        });
        this.setSpellTooltip();
    }
}
```

6.13.4 DivineSword

Listing 53: DivineSword.java

```
package Model.Card.AllCards.Spells.Law;

import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Control.Controllers.PlayerController;
import Model.Card.AllCards.Spells.Spell;
import Model.Card.CardType;
import Model.Card.GenericCardEffect;
import Model.Map.Tile;
import static Resource.Constants.CardConstants.SpellConstants
                                                .Law.DivineSwordSpellConstants.*;
import Resource.Images.Animations;
import Resource.Images.Images;

/**
 * The Divine Sword is a helpful Spell that grants +80 to the Attack of a
 * friendly Wizards when you cast it on him.
 */
public class DivineSword extends Spell
```

```
{

    /**
     * Default Constructor.
     */
    public DivineSword ()
    {
        this.setType(CardType.Law);
        this.setCardName(DIVINE_SWORD_SPELL_CARD_NAME);
        this.setManaCost(DIVINE_SWORD_SPELL_MANA_COST);
        this.setFluxChange(DIVINE_SWORD_SPELL_FLUX_CHANGE);
        this.setID(DIVINE_SWORD_SPELL_CARD_ID);
        this.setCastRange(DIVINE_SWORD_SPELL_CAST_RANGE);
        this.setImage(Images.DIVINE_SWORD);
        this.setDescription(DIVINE_SWORD_SPELL_CARD_DESCRIPTION);
        this.setCardEffect(new GenericCardEffect()
        {
            @Override
            public boolean cast (Tile target)
            {
                GameController.getInstance().getCurrentTurnController()
                    .getCurrentPlayer().getCharacterInstance()
                                .addAttack(DIVINE_SWORD_SPELL_ON_CAST_ADD_ATTACK);
                AnimationController.getInstance()
                                    .playAnimationOnTile(Animations.BUFF, target);
                return true;
            }

            @Override
            public boolean cast (Tile target, int playerID)
            {
                PlayerController.getInstance().getPlayer(playerID)
                    .getCharacterInstance()
                                .addAttack(DIVINE_SWORD_SPELL_ON_CAST_ADD_ATTACK);
                AnimationController.getInstance()
                                    .playAnimationOnTile(Animations.BUFF, target);
                return true;
            }

        });
        this.setSpellTooltip();
    }
}
```

6.14 Model.Card.AllCards.Spells.Neutral

6.14.1 MagicBolt

Listing 54: MagicBolt.java

```
package Model.Card.AllCards.Spells.Neutral;

import Model.Card.AllCards.Spells.AttackSpell;
import Model.Card.CardType;
import static Resource.Constants.CardConstants.SpellConstants.Neutral.MagicBoltSpellConstants.*;
import Resource
                                                        .Images.Images;

/**
 * Magic Bolt is a powerful Spell with a minimum damage of 50 or more when it is
 * cast on a hostile Creature or Wizard.
 */
public class MagicBolt extends AttackSpell
{
```

```
    /**
     * Default Constructor.
     */
    public MagicBolt ()
    {
        this.setType(CardType.Neutral);
        this.setCardName(MAGIC_BOLT_SPELL_CARD_NAME);
        this.setCastRange(MAGIC_BOLT_SPELL_CAST_RANGE);
        this.setID(MAGIC_BOLT_SPELL_CARD_ID);
        this.setImage(Images.MAGIC_BOLT);
        this.setRequiresLineOfSight(MAGIC_BOLT_SPELL_REQUIRES_LINE_OF_SIGHT);
        this.setDescription(MAGIC_BOLT_SPELL_CARD_DESCRIPTION);
        ((AttackSpellEffect) this.getCardEffect()).setAttackValue(
                                MAGIC_BOLT_SPELL_CARD_EFFECT_SET_ATTACK_VALUE);
        ((AttackSpellEffect) this.getCardEffect()).setSpellName(
                                  MAGIC_BOLT_SPELL_CARD_EFFECT_SET_SPELL_NAME);
        this.setSpellTooltip();
    }

}
```

6.14.2 WindWalker

Listing 55: WindWalker.java

```
package Model.Card.AllCards.Spells.Neutral;

import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Control.Controllers.PlayerController;
import Model.Card.AllCards.Spells.Spell;
import Model.Card.CardType;
import Model.Card.GenericCardEffect;
import Model.GameObject.Attribute;
import Model.GameObject.PlayerCharacter;
import Model.Map.Tile;
import static Resource.Constants.CardConstants.SpellConstants.Neutral
                                                .WindWalkerSpellConstants.*;
import Resource.Images.Animations;
import Resource.Images.Images;

/**
 * Wind Walker is a helpful Spell that grants the friendly Wizard where it is
 * casted on +1 movement point and the ability to fly.
 */
public class WindWalker extends Spell
{

    /**
     * Default Constructor.
     */
    public WindWalker ()
    {
        this.setType(CardType.Neutral);
        this.setCardName(WIND_WALKER_SPELL_CARD_NAME);
        this.setManaCost(WIND_WALKER_SPELL_MANA_COST);
        this.setFluxChange(WIND_WALKER_SPELL_FLUX_CHANGE);
        this.setID(WIND_WALKER_SPELL_CARD_ID);
        this.setCastRange(WIND_WALKER_SPELL_CAST_RANGE);
        this.setImage(Images.WINDWALKER);
        this.setDescription(WIND_WALKER_SPELL_CARD_DESCRIPTION);
        this.setCardEffect(new GenericCardEffect()
        {
            @Override
            public boolean cast (Tile target)
            {
```

```
                PlayerCharacter player = GameController.getInstance()
                    .getCurrentTurnController().getCurrentPlayer()
                                                .getCharacterInstance();
                player.setMaxMvp(player.getMaxMvp() +
                                        WIND_WALKER_SPELL_ON_CAST_ADD_MAX_MVP);
                player.addAttribute(Attribute.Flying);
                AnimationController.getInstance().playAnimationOnTile(
                                                Animations.BUFF, target);
                return true;
            }

            @Override
            public boolean cast (Tile target, int playerID)
            {
                PlayerCharacter player = PlayerController.getInstance()
                                    .getPlayer(playerID).getCharacterInstance();
                player.setMaxMvp(player.getMaxMvp() +
                                        WIND_WALKER_SPELL_ON_CAST_ADD_MAX_MVP);
                player.addAttribute(Attribute.Flying);
                AnimationController.getInstance().playAnimationOnTile(
                                                Animations.BUFF, target);
                return true;
            }

        });
        this.setSpellTooltip();
    }

}
```

6.15 Model.Card.AllCards.Summons

6.15.1 Summon

Listing 56: Summon.java

```
package Model.Card.AllCards.Summons;

import Control.Controllers.GameController;
import Control.Controllers.PlayerController;
import Model.Card.CardAdapter;
import Model.Card.CardEffectAdapter;
import Model.GameObject.SummonableMinion;
import Model.Map.Tile;
import static Resource.Constants.CardConstants.SummonConstants.SummonConstants
              .SUMMON_EFFECT_CARD_NAME;
import View.Tooltip.CardTooltip;

/**
 * A Summon is a Creature that is summoned with a Card on a Tile to help
 * the Wizard defeat the other Player.
 */
public class Summon extends CardAdapter
{
    // DO NOT DELETE THIS CLASS!
    // ALL cards that summon minions are subclasses of this class.

    /**
     * Setter for the Tooltip of a the Summon Creature, is set automatically.
     */
    protected void setSummonTooltip ()
    {
        this.setTooltip(new CardTooltip(this.getCardName(), this));
    }
```

```
    /**
     * Sets The Minion that should be summoned upon the card being played.
     *
     * @param minion This Card will summon this Minion.
     */
    protected void setSummon (SummonableMinion minion)
    {
        this.setCardEffect(new SummonEffect(minion));
    }

    /**
     * The SummonEffect is a Cardeffect that summons a minion on the tile where
     * it
     * is casted on.
     */
    private class SummonEffect extends CardEffectAdapter
    {

        /**
         * The Minion that is summoned with this card.
         */
        private final SummonableMinion MINION;

        /**
         * Default Constructor.
         *
         * @param summon - A Instance of the Minion that is summoned by this
         *               card.
         */
        public SummonEffect (SummonableMinion summon)
        {
            this.setName(summon.getName() + SUMMON_EFFECT_CARD_NAME);
            this.MINION = summon;
        }

        /**
         * The Cast Effect is the same for all cards that summon minions,
         * Summons a
         * minion on the target tile.
         *
         * @param target - the target tile.
         *
         * @return - true if the cast was successful.
         */
        @Override
        public boolean cast (Tile target)
        {
            if (this.MINION.summon(target))
              {
                this.MINION.setOwner(GameController.getInstance()
                                .getCurrentTurnController().getCurrentPlayer());
                return true;
              }
            return false;
        }

        /**
         * The Cast Effect is the same for all cards that summon minions,
         * Summons a
         * minion on the target tile and sets it owner to the given playerID.
         *
         * @param target   - the target tile.
         * @param playerID - the id of the player that owns the summoned minion
         *
         * @return - true if the cast was successful.
         */
```

```
        @Override
        public boolean cast (Tile target, int playerID)
        {
            if (this.MINION.summon(target))
              {
                this.MINION.setOwner(PlayerController.getInstance()
                                                        .getPlayer(playerID));
                return true;
              }
            return false;
        }

    }

}
```

6.15.2 SummonEffect

Listing 57: SummonEffect.java

```
package Model.Card.AllCards.Summons;

import Control.Controllers.GameController;
import Control.Controllers.PlayerController;
import Model.Card.CardEffectAdapter;
import Model.GameObject.SummonableMinion;
import Model.Map.Tile;
import static Resource.Constants.CardConstants.SummonConstants.SummonConstants
                                                    .SUMMON_EFFECT_CARD_NAME;

/**
 * The SummonEffect is a Cardeffect that summons a minion on the tile where it
 * is casted on.
 */
public class SummonEffect extends CardEffectAdapter
{

    /**
     * The Minion that is summoned with this card.
     */
    private final SummonableMinion _MINION;

    /**
     * Default Constructor.
     *
     * @param summon - A Instance of the Minion that is summoned by this card.
     */
    public SummonEffect (SummonableMinion summon)
    {
        this.setName(summon.getName() + SUMMON_EFFECT_CARD_NAME);
        this._MINION = summon;
    }

    /**
     * The Cast Effect is the same for all cards that summon minions, Summons a
     * minion on the target tile.
     *
     * @param target - the target tile.
     *
     * @return - true if the cast was successful.
     */
    @Override
    public boolean cast (Tile target)
    {
        if (this._MINION.summon(target))
          {
```

```
            this._MINION.setOwner(GameController.getInstance()
                                  .getCurrentTurnController().getCurrentPlayer());
            return true;
          }
        return false;
    }

    /**
     * The Cast Effect is the same for all cards that summon minions, Summons a
     * minion on the target tile and sets it owner to the given playerID.
     *
     * @param target - the target tile.
     * @param playerID - the id of the player that owns the summoned minion
     * @return - true if the cast was succesful.
     */
    @Override
    public boolean cast (Tile target, int playerID)
    {
        if (this._MINION.summon(target))
          {
            this._MINION.setOwner(PlayerController.getInstance()
                                                        .getPlayer(playerID));
            return true;
          }
        return false;
    }

}
```

6.16 Model.Card.AllCards.Summons.Chaos

6.16.1 Hellhound

Listing 58: Hellhound.java

```
package Model.Card.AllCards.Summons.Chaos;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Chaos.HellhoundMinion;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.HellhoundSummonConstants.*;
import Resource
                                                    .Images.Images;

/**
 * The Hellhound is a Creature that is summoned by a Card and has a FluxChange
 * of 4 to Chaos.
 */
public class Hellhound extends Summon
{

    /**
     * Default Constructor.
     */
    public Hellhound ()
    {
        this.setType(CardType.Chaos);
        this.setImage(Images.HELLHOUND);
        this.setCardName(HELLHOUND_SUMMON_CARD_NAME);
        this.setManaCost(HELLHOUND_SUMMON_MANA_COST);
        this.setFluxChange(HELLHOUND_SUMMON_FLUX_CHANGE);
        this.setDescription(HELLHOUND_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new HellhoundMinion());
        this.setID(HELLHOUND_SUMMON_CARD_ID);
        super.setSummonTooltip();
```

```
    }

}
```

6.16.2 Hydra

Listing 59: Hydra.java

```
package Model.Card.AllCards.Summons.Chaos;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Chaos.HydraMinion;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.HydraSummonConstants.*;
import Resource
                                                        .Images.Images;

/**
 * The Hydra is a summoned Creature that has Fluxchange of 5 to Chaos.
 */
public class Hydra extends Summon
{

    /**
     * Default Constructor.
     */
    public Hydra ()
    {
        this.setType(CardType.Chaos);
        this.setImage(Images.HYDRA);
        this.setCardName(HYDRA_SUMMON_CARD_NAME);
        this.setManaCost(HYDRA_SUMMON_MANA_COST);
        this.setFluxChange(HYDRA_SUMMON_FLUX_CHANGE);
        this.setID(HYDRA_SUMMON_CARD_ID);
        this.setDescription(HYDRA_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new HydraMinion());
        super.setSummonTooltip();
    }

}
```

6.16.3 Manticore

Listing 60: Manticore.java

```
package Model.Card.AllCards.Summons.Chaos;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Chaos.ManticoreMinion;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.ManticoreSummonConstants.*;
import Resource
                                                        .Images.Images;

/**
 * The Manticore is a summoned Creature that has a FluxChange of 4 to Chaos.
 */
public class Manticore extends Summon
{

    /**
     * Default Constructor.
     */
    public Manticore ()
    {
```

```
        this.setType(CardType.Chaos);
        this.setCardName(MANTICORE_SUMMON_CARD_NAME);
        this.setImage(Images.MANTICORE);
        this.setManaCost(MANTICORE_SUMMON_MANA_COST);
        this.setDescription(MANTICORE_SUMMON_CARD_DESCRIPTION);
        this.setFluxChange(MANTICORE_SUMMON_FLUX_CHANGE);
        this.setID(MANTICORE_SUMMON_CARD_ID);
        this.setSummon(new ManticoreMinion());
        super.setSummonTooltip();
    }

}
```

6.16.4 Skeleton

Listing 61: Skeleton.java

```
package Model.Card.AllCards.Summons.Chaos;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Chaos.SkeletonMinion;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.SkeletonSummonConstants.*;
import Resource
                                                        .Images.Images;

/**
 * The Skeleton is a summoned Creature that has a Fluxchange of 3.
 */
public class Skeleton extends Summon
{

    /**
     * Default Constructor.
     */
    public Skeleton ()
    {
        this.setType(CardType.Chaos);
        this.setCardName(SKELETON_SUMMON_CARD_NAME);
        this.setImage(Images.SKELETON);
        this.setManaCost(SKELETON_SUMMON_MANA_COST);
        this.setID(SKELETON_SUMMON_CARD_ID);
        this.setDescription(SKELETON_SUMMON_CARD_DESCRIPTION);
        this.setFluxChange(SKELETON_SUMMON_FLUX_CHANGE);
        this.setSummon(new SkeletonMinion());
        super.setSummonTooltip();
    }
}
```

6.16.5 Vampire

Listing 62: Vampire.java

```
package Model.Card.AllCards.Summons.Chaos;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Chaos.VampireMinion;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.VampireSummonConstants.*;
import Resource
                                                        .Images.Images;

/**
 * Vampire is a Undead Creature that has a Fluxchange of 4.
 */
```

```
public class Vampire extends Summon
{

    /**
     * Default Constructor.
     */
    public Vampire ()
    {
        this.setType(CardType.Chaos);
        this.setCardName(VAMPIRE_SUMMON_CARD_NAME);
        this.setImage(Images.VAMPIRE);
        this.setManaCost(VAMPIRE_SUMMON_MANA_COST);
        this.setFluxChange(VAMPIRE_SUMMON_FLUX_CHANGE);
        this.setID(VAMPIRE_SUMMON_CARD_ID);
        this.setDescription(VAMPIRE_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new VampireMinion());
        super.setSummonTooltip();
    }

}
```

6.16.6 Zombie

Listing 63: Zombie.java

```
package Model.Card.AllCards.Summons.Chaos;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Chaos.ZombieMinion;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.ZombieSummonConstants.*;
import Resource
                                                        .Images.Images;

/**
 * Zombie is a summon undead Creature that has a Fluxchange of 2 to Chaos.
 */
public class Zombie extends Summon
{

    /**
     * Default Constructor.
     */
    public Zombie ()
    {
        this.setType(CardType.Chaos);
        this.setCardName(ZOMBIE_SUMMON_CARD_NAME);
        this.setImage(Images.ZOMBIE);
        this.setManaCost(ZOMBIE_SUMMON_MANA_COST);
        this.setFluxChange(ZOMBIE_SUMMON_FLUX_CHANGE);
        this.setID(ZOMBIE_SUMMON_CARD_ID);
        this.setDescription(ZOMBIE_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new ZombieMinion());
        super.setSummonTooltip();
    }

}
```

6.17 Model.Card.AllCards.Summons.Law

6.17.1 IcarusTower

Listing 64: IcarusTower.java

```
package Model.Card.AllCards.Summons.Law;
```

```
import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Law.IcarusTowerMinion;
import static Resource.Constants.CardConstants.SummonConstants.Law.IcarusTowerSummonConstants.*;
import Resource
                                                    .Images.Images;

/**
 * Icarus Tower is a summon Creature that has a Fluxchange of 3 to Law.
 */
public class IcarusTower extends Summon
{

    /**
     * Default Constructor.
     */
    public IcarusTower ()
    {
        this.setType(CardType.Law);
        this.setCardName(ICARUS_TOWER_SUMMON_CARD_NAME);
        this.setImage(Images.ICARUS_TOWER);
        this.setManaCost(ICARUS_TOWER_SUMMON_MANA_COST);
        this.setID(ICARUS_TOWER_SUMMON_CARD_ID);
        this.setFluxChange(ICARUS_TOWER_SUMMON_FLUX_CHANGE);
        this.setDescription(ICARUS_TOWER_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new IcarusTowerMinion());
        super.setSummonTooltip();
    }

}
```

6.17.2 Paladin

Listing 65: Paladin.java

```
package Model.Card.AllCards.Summons.Law;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Law.PaladinMinion;
import static Resource.Constants.CardConstants.SummonConstants.Law.PaladinSummonConstants.*;
import Resource
                                                        .Images.Images;

/**
 * Paladin is summon Creature that has a Fluxchange of 4 to Law.
 */
public class Paladin extends Summon
{

    /**
     * Default Constructor.
     */
    public Paladin ()
    {
        this.setType(CardType.Law);
        this.setCardName(PALADIN_SUMMON_CARD_NAME);
        this.setImage(Images.PALADIN);
        this.setManaCost(PALADIN_SUMMON_MANA_COST);
        this.setFluxChange(PALADIN_SUMMON_FLUX_CHANGE);
        this.setID(PALADIN_SUMMON_CARD_ID);
        this.setDescription(PALADIN_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new PaladinMinion());
        super.setSummonTooltip();
    }
```

```
}
```

6.17.3 Pegasus

Listing 66: Pegasus.java

```
package Model.Card.AllCards.Summons.Law;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Law.PegasusMinion;
import static Resource.Constants.CardConstants.SummonConstants.Law.PegasusSummonConstants.*;
import Resource
                                                        .Images.Images;

/**
 * Pegasus is a summon creature that has Fluxchange of 4 to Law.
 */
public class Pegasus extends Summon
{

    /**
     * Default Constructor.
     */
    public Pegasus ()
    {
        this.setType(CardType.Law);
        this.setCardName(PALADIN_SUMMON_CARD_NAME);
        this.setImage(Images.PEGASUS);
        this.setManaCost(PALADIN_SUMMON_MANA_COST);
        this.setFluxChange(PALADIN_SUMMON_FLUX_CHANGE);
        this.setID(PEGASUS_SUMMON_CARD_ID);
        this.setDescription(PALADIN_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new PegasusMinion());
        super.setSummonTooltip();
    }

}
```

6.17.4 SapphireDragon

Listing 67: SapphireDragon.java

```
package Model.Card.AllCards.Summons.Law;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Law.SapphireDragonMinion;
import static Resource.Constants.CardConstants.SummonConstants.Law
                                                    .SapphireDragonSummonConstants.*;
import Resource.Images.Images;

/**
 * SapphireDragon is a summon Creeature that has a Fluxchange of 5 to law.
 */
public class SapphireDragon extends Summon
{

    /**
     * Default Constructor.
     */
    public SapphireDragon ()
    {
        this.setType(CardType.Law);
```

```
        this.setCardName(SAPPHIRE_DRAGON_SUMMON_CARD_NAME);
        this.setImage(Images.SAPPHIRE_DRAGON);
        this.setManaCost(SAPPHIRE_DRAGON_SUMMON_MANA_COST);
        this.setFluxChange(SAPPHIRE_DRAGON_SUMMON_FLUX_CHANGE);
        this.setDescription(SAPPHIRE_DRAGON_SUMMON_CARD_DESCRIPTION);
        this.setID(SAPPHIRE_DRAGON_SUMMON_CARD_ID);
        this.setSummon(new SapphireDragonMinion());
        super.setSummonTooltip();
    }

}
```

6.18 Model.Card.AllCards.Summons.Neutral

6.18.1 AirElemental

Listing 68: AirElemental.java

```
package Model.Card.AllCards.Summons.Neutral;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Neutral.AirElementalMinion;
import static Resource.Constants.CardConstants.SummonConstants.Neutral
                                                .AirElementalSummonConstants.*;
import Resource.Images.Images;

/**
 * Air Elemental is a summon Creature with no Fluxchange.
 */
public class AirElemental extends Summon
{

    /**
     * Default Constructor.
     */
    public AirElemental ()
    {
        this.setType(CardType.Neutral);
        this.setCardName(AIR_ELEMENTAL_SUMMON_CARD_NAME);
        this.setImage(Images.AIR_ELEMENTAL);
        this.setID(AIR_ELEMENTAL_SUMMON_CARD_ID);
        this.setManaCost(AIR_ELEMENTAL_SUMMON_MANA_COST);
        this.setDescription(AIR_ELEMENTAL_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new AirElementalMinion());
        super.setSummonTooltip();
    }
}
```

6.18.2 Eagle

Listing 69: Eagle.java

```
package Model.Card.AllCards.Summons.Neutral;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Neutral.EagleMinion;
import static Resource.Constants.CardConstants.SummonConstants.Neutral
                                                    .EagleSummonConstants.*;
import Resource.Images.Images;

/**
 * Eagle is a summon Creature with no Fluxchange.
```

```
 */
public class Eagle extends Summon
{

    /**
     * Default Constructor.
     */
    public Eagle ()
    {
        this.setType(CardType.Neutral);
        this.setCardName(EAGLE_SUMMON_CARD_NAME);
        this.setImage(Images.EAGLE);
        this.setID(EAGLE_SUMMON_CARD_ID);
        this.setManaCost(EAGLE_SUMMON_MANA_COST);
        this.setDescription(EAGLE_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new EagleMinion());
        super.setSummonTooltip();
    }

}
```

6.18.3 Giant

Listing 70: Giant.java

```
package Model.Card.AllCards.Summons.Neutral;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
import Model.GameObject.Minions.Neutral.GiantMinion;
import static Resource.Constants.CardConstants.SummonConstants.Neutral
                                                        .GiantSummonConstants.*;
import Resource.Images.Images;

/**
 * Giant is a summon Creature with no FLuxchange.
 */
public class Giant extends Summon
{

    /**
     * Default Constructor.
     */
    public Giant ()
    {
        this.setType(CardType.Neutral);
        this.setCardName(GIANT_SUMMON_CARD_NAME);
        this.setImage(Images.GIANT);
        this.setID(GIANT_SUMMON_CARD_ID);
        this.setManaCost(GIANT_SUMMON_MANA_COST);
        this.setDescription(GIANT_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new GiantMinion());
        super.setSummonTooltip();
    }

}
```

6.18.4 Lion

Listing 71: Lion.java

```
package Model.Card.AllCards.Summons.Neutral;

import Model.Card.AllCards.Summons.Summon;
import Model.Card.CardType;
```

```
import Model.GameObject.Minions.Neutral.LionMinion;
import static Resource.Constants.CardConstants.SummonConstants.Neutral
                                                        .LionSummonConstants.*;
import static Resource.Constants.CardConstants.SummonConstants.SummonConstants
                                                          .DEFAULT_FLUX_CHANGE;
import Resource.Images.Images;

/**
 * Lion is a summon Creature with no Fluxchange.
 */
public class Lion extends Summon
{

    /**
     * Default Constructor.
     */
    public Lion ()
    {
        this.setType(CardType.Neutral);
        this.setCardName(LION_SUMMON_CARD_NAME);
        this.setImage(Images.LION);
        this.setID(LION_SUMMON_CARD_ID);
        this.setManaCost(LION_SUMMON_MANA_COST);
        this.setFluxChange(DEFAULT_FLUX_CHANGE);
        this.setDescription(LION_SUMMON_CARD_DESCRIPTION);
        this.setSummon(new LionMinion());
        super.setSummonTooltip();
    }

}
```

6.19 Model.Card.Wands

6.19.1 AirWand

Listing 72: AirWand.java

```
package Model.Card.Wands;

import Model.Card.Wand;
import static Resource.Constants.CardConstants.SpellConstants.Neutral.MagicBoltSpellConstants.
        MAGIC_BOLT_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SpellConstants.Neutral.WindWalkerSpellConstants.
        WIND_WALKER_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.ManticoreSummonConstants.
        MANTICORE_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.VampireSummonConstants.
        VAMPIRE_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Law.IcarusTowerSummonConstants.
        ICARUS_TOWER_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Law.PegasusSummonConstants.
        PEGASUS_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Law.SapphireDragonSummonConstants.
        SAPPHIRE_DRAGON_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Neutral.AirElementalSummonConstants.
        AIR_ELEMENTAL_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Neutral.EagleSummonConstants.
        EAGLE_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.WandConstants.AirWandConstants.
        *;
import static Resource.Images.WandImages.AIR_WAND;

/**
 * The Airwand is a Wand with cards that are related to the Element of Air,
 * some minions that can be summoned can fly, others can attack flying
```

```
 * creatures and some cards have effects like nolts.
 */
public class AirWand extends Wand
{

    /**
     * Default Constructor.
     */
    public AirWand ()
    {
        super(AIR_WAND_NAME);
    }

    /**
     * Adds all cards which this Wand provides.
     */
    @Override
    public void init ()
    {
        this.setImage(AIR_WAND);

        this.addCards(SAPPHIRE_DRAGON_SUMMON_CARD_ID,
                AMOUNT_OF_SAPPHIRE_DRAGON_AIR_WAND);
        this.addCards(PEGASUS_SUMMON_CARD_ID,
                AMOUNT_OF_PEGASUS_AIR_WAND);
        this.addCards(VAMPIRE_SUMMON_CARD_ID,
                AMOUNT_OF_VAMPIRE_AIR_WAND);
        this.addCards(MANTICORE_SUMMON_CARD_ID, AMOUNT_OF_MANTICORE_AIR_WAND);
        this.addCards(EAGLE_SUMMON_CARD_ID, AMOUNT_OF_EAGLE_AIR_WAND);
        this.addCards(ICARUS_TOWER_SUMMON_CARD_ID,
                AMOUNT_OF_ICARUS_TOWER_AIR_WAND);
        this.addCards(WIND_WALKER_SPELL_CARD_ID,
                AMOUNT_OF_WIND_WALKER_AIR_WAND);
        this.addCards(AIR_ELEMENTAL_SUMMON_CARD_ID,
                AMOUNT_OF_AIR_ELEMENTAL_AIR_WAND);
        this.addCards(MAGIC_BOLT_SPELL_CARD_ID, AMOUNT_OF_MAGIC_BOLT_AIR_WAND);
    }

}
```

6.19.2 CreatureWand

Listing 73: CreatureWand.java

```
package Model.Card.Wands;

import Model.Card.Wand;
import static Resource.Constants.CardConstants.SpellConstants.Chaos.
        VengeanceSpellConstants.VENGEANCE_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SpellConstants.Law.
        DecreeSpellConstants.DECREE_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SpellConstants.Law.
        DivineBowSpellConstants.DIVINE_BOW_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SpellConstants.Law.
        DivineShieldSpellConstants.DIVINE_SHIELD_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SpellConstants.Law.
        DivineSwordSpellConstants.DIVINE_SWORD_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SpellConstants.Neutral.
        MagicBoltSpellConstants.MAGIC_BOLT_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.
        HydraSummonConstants.HYDRA_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.
        VampireSummonConstants.VAMPIRE_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Law.
        PaladinSummonConstants.PALADIN_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Law.
```

```
        SapphireDragonSummonConstants.SAPPHIRE_DRAGON_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Neutral.
        AirElementalSummonConstants.AIR_ELEMENTAL_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Neutral.
        GiantSummonConstants.GIANT_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Neutral.
        LionSummonConstants.LION_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.WandConstants.
        CreatureWandConstants.*;
import Resource.Images.Images;

/**
 * The CreatureWand is a Wand that contains cards that can summon creatures or
 * have helpful spell for creatures.
 */
public class CreatureWand extends Wand
{

    /**
     * Default Constructor.
     */
    public CreatureWand ()
    {
        super(CREATURE_WAND_NAME);
    }

    /**
     * Adds all cards which this Wand provides.
     */
    @Override
    public void init ()
    {
        this.setImage(Images.CREATURE_WAND);

        this.addCards(SAPPHIRE_DRAGON_SUMMON_CARD_ID,
                AMOUNT_OF_SAPPHIRE_DRAGON_CREATURE_WAND);
        this.addCards(PALADIN_SUMMON_CARD_ID, AMOUNT_OF_PALADIN_CREATURE_WAND);
        this.addCards(GIANT_SUMMON_CARD_ID, AMOUNT_OF_GIANT_CREATURE_WAND);
        this.addCards(LION_SUMMON_CARD_ID, AMOUNT_OF_LION_CREATURE_WAND);
        this.addCards(HYDRA_SUMMON_CARD_ID, AMOUNT_OF_HYDRA_CREATURE_WAND);
        this.addCards(VAMPIRE_SUMMON_CARD_ID, AMOUNT_OF_VAMPIRE_CREATURE_WAND);
        this.addCards(DIVINE_BOW_SPELL_CARD_ID,
                AMOUNT_OF_DIVINE_BOW_CREATURE_WAND);
        this.addCards(DIVINE_SHIELD_SPELL_CARD_ID,
                AMOUNT_OF_DIVINE_SHIELD_CREATURE_WAND);
        this.addCards(DIVINE_SWORD_SPELL_CARD_ID,
                AMOUNT_OF_DIVINE_SWORD_CREATURE_WAND);
        this.addCards(DECREE_SPELL_CARD_ID,
                AMOUNT_OF_DECREE_CREATURE_WAND);
        this.addCards(VENGEANCE_SPELL_CARD_ID,
                AMOUNT_OF_VENGEANCE_CREATURE_WAND);
        this.addCards(MAGIC_BOLT_SPELL_CARD_ID,
                AMOUNT_OF_MAGIC_BOLT_CREATURE_WAND);
        this.addCards(AIR_ELEMENTAL_SUMMON_CARD_ID,
                AMOUNT_OF_AIR_ELEMENTAL_CREATURE_WAND);
    }

}
```

6.19.3 RandomWand

Listing 74: RandomWand.java

```
package Model.Card.Wands;

import Model.Card.Wand;
```

```
import Model.Card.WandFactory;
import static Resource.Constants.CardConstants.WandConstants.RandomWandConstants
        .*;
import static Resource.Images.WandImages
        .RANDOM_WAND;

/**
 * The RandomWand is a randomly generated wand with random cards.
 */
public class RandomWand extends Wand
{

    /**
     * Default Constructor.
     */
    public RandomWand ()
    {
        super(RANDOM_WAND_NAME);
    }

    /**
     * Initializes the random Wand with radnom Cards.
     */
    @Override
    public void init ()
    {
        this.setImage(RANDOM_WAND);

        this.getTooltip().addInfo(RANDOM_WAND_TOOLTIP_INFO_WAND,
                RANDOM_WAND_TOOLTIP_INFO_RANDOM);

        this.setCards(WandFactory.getRandomWand().getCards());
    }

}
```

6.19.4 UndeadWand

Listing 75: UndeadWand.java

```
package Model.Card.Wands;

import Model.Card.Wand;
import static Resource.Constants.CardConstants.SpellConstants.Chaos.
        VengeanceSpellConstants.VENGEANCE_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SpellConstants.Law.
        DecreeSpellConstants.DECREE_SPELL_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.
        HellhoundSummonConstants.HELLHOUND_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.
        SkeletonSummonConstants.SKELETON_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.
        VampireSummonConstants.VAMPIRE_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Chaos.
        ZombieSummonConstants.ZOMBIE_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.SummonConstants.Neutral.
        LionSummonConstants.LION_SUMMON_CARD_ID;
import static Resource.Constants.CardConstants.WandConstants.
        UndeadWandConstants.*;
import Resource.Images.Images;

/**
 * The Undead Wand is a Wand that contains cards with the skil of being
 * undead, like Zombies or Vampires.
 */
public class UndeadWand extends Wand
```

```
{

    /**
     * Default Constructor.
     */
    public UndeadWand ()
    {
        super(UNDEAD_WAND_NAME);
    }

    /**
     * Adds all cards which this Wand provides.
     */
    @Override
    public void init ()
    {
        this.setImage(Images.UNDEAD_WAND);

        this.addCards(LION_SUMMON_CARD_ID, AMOUNT_OF_LION_UNDEAD_WAND);
        this.addCards(VAMPIRE_SUMMON_CARD_ID, AMOUNT_OF_VAMPIRE_UNDEAD_WAND);
        this.addCards(SKELETON_SUMMON_CARD_ID, AMOUNT_OF_SKELETON_UNDEAD_WAND);
        this.addCards(HELLHOUND_SUMMON_CARD_ID, AMOUNT_OF_HELLHOUND_UNDEAD_WAND);
        this.addCards(DECREE_SPELL_CARD_ID, AMOUNT_OF_DECREE_UNDEAD_WAND);
        this.addCards(VENGEANCE_SPELL_CARD_ID, AMOUNT_OF_VENGEANCE_UNDEAD_WAND);
        this.addCards(ZOMBIE_SUMMON_CARD_ID, AMOUNT_OF_ZOMBIE_UNDEAD_WAND);
    }

}
```

6.20 Model.Game

6.20.1 FluxMeter

Listing 76: FluxMeter.java

```
package Model.Game;

import static Resource.Constants.GeneralConstants.GameConstants.MAX_FLUX;
import static Resource.Constants.GeneralConstants.GameConstants.MIN_FLUX;

/**
 * The Fluxmeter indicates which side has the most power at the moment, it
 * increases cost for cards for the side where it deflects to, the Fluxmeter
 * can be influenced with playing cards.
 */
public class FluxMeter
{

    /**
     * The amount of flux that is currently in the meter.
     */
    private int _flux = 0;

    /**
     * The maximum of flux that can be in the meter.
     */
    private int _maxFlux = 0;

    /**
     * The minimum of flux that can be in the meter.
     */
    private int _minFlux = 0;

    /**
     * Instance to make the Fluxmeter a Singleton.
```

```
31      */
32     private static FluxMeter _instance = new FluxMeter();

34     /**
35      * Default Constructor.
36      */
37     private FluxMeter ()
38     {
39         this._maxFlux = MAX_FLUX;
40         this._minFlux = MIN_FLUX;
41     }

43     /**
44      * Getter for this classes singleton instance.
45      *
46      * @return - instance of the Fluxmeter.
47      */
48     public static FluxMeter getInstance ()
49     {
50         return _instance;
51     }

53     /**
54      * Use this to get the current amount of flux in the meter.
55      *
56      * @return - the current amount of flux in the meter.
57      */
58     public int getFlux ()
59     {
60         return _flux;
61     }

63     /**
64      * Changes the value of the flux meter according to the given value.
65      *
66      * @param value - negative values to shift towards law, positive values to
67      *               shift towards chaos.
68      */
69     public void addFlux (int value)
70     {
71         if (_flux + value > _maxFlux)
72           {
73             _flux = _maxFlux;
74           }
75         else if (_flux + value < _minFlux)
76           {
77             _flux = _minFlux;
78           }
79         else
80           {
81             _flux += value;
82           }
83     }

85     /**
86      * Getter to find out to which side the fluxmeter deflects.
87      *
88      * @return - 0 if flux is on 0, 1 if flux is less 0, -1 if flux is more than 0
89      */
90     public int getLean ()
91     {
92         if (_flux == 0)
93           {
94             return 0;
95           }
96         if (_flux < 0)
```

```
            {
              return 1;
            }
        return -1;
    }

    /**
     * Resets the Fluxmeter to 0.
     */
    public void init ()
    {
        this._flux = 0;
    }
}
```

6.21 Model.GameObject

6.21.1 Attribute

Listing 77: Attribute.java

```
package Model.GameObject;

/**
 * This Enumeration is a list of all Attributes that are currently Implemented
 * in this Game.
 *
 * Creatures can have any amount of these Attributes, but not more than one of
 * the same Type.
 */
public enum Attribute
{

    /**
     * If the Creature has this Attribute it can traverse Height Differences in
     * Tiles greater than 1.
     */
    Flying,

    /**
     * If the Creature has this Attribute it heals on any Attack and deals Extra
     * Damage that cannot be mitigated by Resistances.
     */
    Leech,

    /**
     * If the Creature has this Attribute it automatically attacks Flying
     * Creatures that pass through a Tile in Range of this Creature.
     */
    AirInterception,

    /**
     * If the Creature has this Attribute it cannot be targeted by Spells.
     */
    Structure,

    /**
     * If the Creature has this Attribute it cannot traverse Tile Height
     * Differences.
     */
    FixedLevel,

    /**
     * If the Creature has this Attribute it knocks its Target back by one tile
     * on any Attack.
```

```
     */
    Knockback,

    /**
     * If the Creature has this Attribute its Ranged Attacks immobilize its
     * Target for one Turn.
     */
    Paralysing,

    /**
     * If the Creature has this Attribute it can only be attacked by other
     * Undead Creatures or Creatures with the Attribute "UndeadSlayer".
     */
    Undead,

    /**
     * If the Creature has this Attribute it is able to attack Creatures with
     * the "Undead" Attribute.
     */
    UndeadSlayer,

    /**
     * If the Creature has this Attribute, it is Paralysed (See Attribute
     * "Paralysing").
     */
    Paralyzed,

    /**
     * If the Creature has this Attribute it is able to perform a Ranged Attack,
     * but only once.
     */
    OneTimeShot
}
```

6.21.2 CreatureAdapter

Listing 78: CreatureAdapter.java

```
package Model.GameObject;

import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Control.Controllers.MapController;
import Model.Player;
import Resource.Constants.GeneralConstants.GameObjectConstants;
import static Resource.Constants.GeneralConstants.GameObjectConstants
                                                        .DEFAULT_ATTACK_RANGE;
import Resource.Images.Animations;
import java.util.ArrayList;
import java.util.List;

/**
 * All Creatures are supposed to Inherit from this Class.
 */
public abstract class CreatureAdapter extends GameObject implements IAttackable
{

    /**
     * Gets set to true if the Creature gets Air Intercepted while moving, so
     * that it cannot be air intercepted a second time in one move action.
     */
    private boolean _airInterceptionAttack = false;

    /**
     * The ID of the Owner Player.
     */
```

```
    private int _owner = 0;

    /**
     * The Creatures Base Attack Value. Used in Melee Attacks.
     */
    private int _attack = 0;

    /**
     * The Creatures Base Ranged Attack Value. Used in Attacks From further than
     * 1 Tile away.
     */
    private int _rangedAttack = 0;

    /**
     * The Creatures Attack Range. if 1, the Creature will only perform Melee
     * Attacks. if higher, the Creature is able to perform Ranged Attacks.
     */
    private int _attackRange = 0;

    /**
     * The Creatures Remaining Movement Points. A Creature is only able to move
     * if this value is >= 1.
     */
    private int _movementpoints = 0;

    /**
     * The Creatures Maximum Movement Points. The Creatures Movement Points will
     * be set to this Value at the start of a new Turn.
     */
    private int _maxMvP = 0;

    /**
     * The Creatures Current HP Value. A Creature dies once this reaches 0.
     */
    private int _health = 0;

    /**
     * The Creatures Maximum Health Value. A Creature cannot be Healed above
     * this HP Value.
     */
    private int _maxHealth = 0;

    /**
     * The Creatures Resistance to Physical (Melee or Ranged) Attacks. The
     * Creature will take this Value as a Percentage of the incoming Attack less
     * Damage.
     */
    private int _physRes = 0;

    /**
     * The Creatures Resistance to Magical (Spells) Attacks. The Creature will
     * take this Value (divided by 2) as a Percentage of the incoming Attack
     * less Damage.
     */
    private int _magicRes = 0;

    /**
     * Remembers if this Creature already attacked in the current Turn. A
     * Creature cannot attack twice per Turn. Set to false at the start of a new
     * Turn.
     */
    private boolean _hasAttacked = false;

    /**
     * Stores this Creatures Attributes (e.g. Flying, Undead, ...)
     */
```

```
    private List<Attribute> _attributes = null;

    /**
     * All Creatures are supposed to Inherit from this Class. This class is not
     * supposed to be initiated.
     */
    public CreatureAdapter ()
    {
        this._attributes = new ArrayList<Attribute>();
    }

    /**
     * Getter for this Creatures Attack Range Value.
     *
     * @return This Creatures Attack Range.
     */
    public int getAttackRange ()
    {
        return this._attackRange;
    }

    /**
     * Setter for this Creatures Attack Range Value.
     *
     * @param attackRange Sets this Creatures Attack Range to the given Value.
     */
    public void setAttackRange (int attackRange)
    {
        this._attackRange = attackRange;
    }

    /**
     * Getter for this Creatures Attack Value.
     *
     * @return This Creatures Base Attack Damage.
     */
    public int getAttack ()
    {
        return this._attack;
    }

    /**
     * Setter for this Creatures Attack Value-
     *
     * @param attack This Creatures Base Attack Damage Value will be set to the
     *               given Value.
     */
    public void setAttack (int attack)
    {
        this._attack = attack;
    }

    /**
     * Getter for this Creatures Ranged Attack Value.
     *
     * @return This Creatures Base Ranged Attack Damage.
     */
    public int getRangedAttack ()
    {
        return this._rangedAttack;
    }

    /**
     * Setter for this Creatures Base Ranged Attack Damage.
     *
     * @param rangedAttack The Creatures Base Ranged Attack Damage Value will be
```

```
     *                     set to the given Value.
     */
    public void setRangedAttack (int rangedAttack)
    {
        this._rangedAttack = rangedAttack;
    }

    /**
     * Getter for this Creatures Maximum Movement Point Value.
     *
     * @return This Creatures Maximum Movement Points.
     */
    public int getMaxMvp ()
    {
        return this._maxMvP;
    }

    /**
     * Setter for this Creatures Maximum Movement Point Value.
     *
     * @param maxMvp The Minions Maximum Movement Point Value will be set to the
     *               given Value.
     */
    public void setMaxMvp (int maxMvp)
    {
        this._maxMvP = maxMvp;
    }

    /**
     * Getter for this Creature Maximum Health Value.
     *
     * @return This Creatures Maximum HP.
     */
    public int getMaxHealth ()
    {
        return this._maxHealth;
    }

    /**
     * Setter for this Creatures Maximum Health Value.
     *
     * @param maxHP The Creatures Maximum Health Value will be set to the given
     *              Value.
     */
    protected void setMaxHealth (int maxHP)
    {
        this._maxHealth = maxHP;
    }

    /**
     * Getter for this Creatures Owner ID.
     *
     * @return This Creatures Owner ID Value.
     */
    public int getOwner ()
    {
        return this._owner;
    }

    /**
     * Setter for this Creatures Owner ID.
     *
     * @param ownerPlayer This Creatures Owner ID will be set the the ID of the
     *                    given Player.
     */
    public void setOwner (Player ownerPlayer)
```

```
    {
        this._owner = ownerPlayer.getOwnerNumber();
    }

    /**
     * Setter for this Creatures Owner ID.
     *
     * @param ownerID This Creatures Owner ID will be set to the given Value.
     */
    protected void setOwnerNumber (int ownerID)
    {
        this._owner = ownerID;
    }

    /**
     * Getter for this Creatures Movement Point Value.
     *
     * @return This Creatures current Movement Points.
     */
    public int getMovementpoints ()
    {
        return this._movementpoints;
    }

    /**
     * Setter for this Creatures Movement Point Value.
     *
     * @param movementPoints The Creatures current Movement Points will be set
     *                       to the given Value.
     */
    public void setMovementpoints (int movementPoints)
    {
        this._movementpoints = movementPoints;
    }

    /**
     * Getter for this Creatures Health Value.
     *
     * @return This Creatures current HP.
     */
    public int getHealth ()
    {
        return this._health;
    }

    /**
     * Setter for this Creatures Health Value.
     *
     * @param health This Creature current HP will be set to the given Value.
     */
    protected void setHealth (int health)
    {
        this._health = health;
    }

    /**
     * Getter for this Creatures Physical Resistance Value. Inherited from
     * IAttackable.
     *
     * @return This Creatures Physical Resistance Value.
     */
    @Override
    public int _getPhysRes ()
    {
        return this._physRes;
    }
```

```
    /**
     * Setter for this Creatures Physical Resistance Value.
     *
     * @param physRes The Creatures Physical Resistance Value will be set to the
     *                given Value.
     */
    public void setPhysRes (int physRes)
    {
        this._physRes = physRes;
    }

    /**
     * Setter for this Creatures Air-Interception-Attack-Value.
     *
     * @param airInterceptionAttack This Creatures Air-Interception-Attack-Value
     *                              will be set to the given Value.
     */
    public void setAirInterceptionAttack (boolean airInterceptionAttack)
    {
        this._airInterceptionAttack = airInterceptionAttack;
    }

    /**
     * Getter for this Creatures Magical Resistance Value. Inherited from
     * IAttackable
     *
     * @return This Creatures Magical Resistance Value.
     */
    @Override
    public int _getMagicRes ()
    {
        return this._magicRes;
    }

    /**
     * Setter for this Creatures Magical Resistance Value.
     *
     * @param magicRes This Creatures Magical Resistance Value will be set to
     *                 the given Value.
     */
    public void setMagicRes (int magicRes)
    {
        this._magicRes = magicRes;
    }

    /**
     * Getter for this Creatures Has-Attacked Value.
     *
     * @return This Creatures Has-Attacked Value.
     */
    public boolean hasAttacked ()
    {
        return this._hasAttacked;
    }

    /**
     * Setter for this Creatures Has-Attacked Value.
     *
     * @param hasAttacked This Creatures Has-Attacked Value will be set to the
     *                    given Value.
     */
    public void setHasAttacked (boolean hasAttacked)
    {
        this._hasAttacked = hasAttacked;
    }
```

```
    /**
     * Resets a Creatures current Movement Points to this Creatures maximum
     * Movement Points. Called at the start of a new Turn.
     */
    public void resetMovementPoints ()
    {
        this._movementpoints = this._maxMvP;
    }

    /**
     * Initializes this Creatures current Values to its Maximum Values, as well
     * as setting its Owner ID. Called once a Creature is summoned onto the
     * Arena.
     */
    protected void init ()
    {
        this._attackRange = DEFAULT_ATTACK_RANGE;
        this.setHealth(this.getMaxHealth());
        this.setMovementpoints(this.getMaxMvp());
        this.setOwner(GameController.getInstance().getCurrentTurnController()
                                                        .getCurrentPlayer());
    }

    /**
     * Adds a given Amount to this Creatures Physical Resistance Value.
     *
     * @param value This Value will be added to this Creatures Physical
     *              Resistance Value.
     */
    public void addPhysRes (int value)
    {
        this._physRes += value;
    }

    /**
     * Adds a given Amount to this Creatures Magical Resistance Value.
     *
     * @param value This Value will be added to this Creatures magical
     *              Resistance Value.
     */
    public void addMagicRes (int value)
    {
        this._magicRes += value;
    }

    /**
     * Adds a given Amount to this Creatures Attack Damage Value.
     *
     * @param value This Value will be added to this Creatures Attack Damage
     *              Value.
     */
    public void addAttack (int value)
    {
        this._attack += value;
    }

    /**
     * Adds a given Amount to this Creatures Attack Range Value.
     *
     * @param value This Value will be added to this Creatures Attack Range
     *              Value.
     */
    public void addAttackRange (int value)
    {
        this._attackRange += value;
```

```
    }

    /**
     * Adds a given Amount to this Creatures Ranged Attack Damage Value.
     *
     * @param value This Value will be added to this Creatures Ranged Attack
     *              Damage Value.
     */
    public void addRangedAttack (int value)
    {
        this._rangedAttack += value;
    }

    /**
     * Adds a given Amount to this Creatures Maximum HP Value.
     *
     * @param value This Value will be added to this Creatures Maximum HP Value.
     */
    public void addMaxHealth (int value)
    {
        this._maxHealth += value;
        this._health += value;
    }

    /**
     * Getter for this Creatures list of Attributes.
     *
     * @return This Creatures list of Attributes.
     */
    public List<Attribute> getAttributes ()
    {
        return this._attributes;
    }

    /**
     * Adds a given Attribute to this Creatures list of Attributes
     *
     * @param att This Attribute will be added to this Creatures list of
     *            Attributes.
     */
    public void addAttribute (Attribute att)
    {
        if (!this._attributes.contains(att))
          {
            this._attributes.add(att);
          }
    }

    /**
     * Adds multiple given Attributes to this Creatures list of Attributes.
     *
     * @param att These Attributes will be added to this Creatures list of
     *            Attributes.
     */
    public void addAttributes (Attribute[] att)
    {
        for (Attribute at : att)
          {
            this.addAttribute(at);
          }
    }

    /**
     * Removes a given Attribute from this Creatures list of Attributes.
     *
     * @param att This Attribute will be removed from this Creatures list of
```

```
     *             Attributes.
     *
     * @return Returns True if the Attribute could successfully be removed.
     */
    public boolean removeAttribute (Attribute att)
    {
        if (!this.hasAttribute(att))
          {
            return false;
          }
        this._attributes.remove(att);
        return true;
    }

    /**
     * Checks if the Creature has the given Attribute in its list of Attributes.
     *
     * @param att The given Attribute.
     *
     * @return True if the Creatures Attribute list contains the given
     *         Attribute.
     */
    public boolean hasAttribute (Attribute att)
    {
        return (this._attributes.contains(att));
    }

    /**
     * Use this to check if the creature is owned by the given player.
     *
     * @param p the player of whom it is unclear of whether he owns this
     *          creature.
     *
     * @return true if this creature is owned by the given player.
     */
    public boolean isOwnedBy (Player p)
    {
        return (this._owner == p.getOwnerNumber());
    }

    /**
     * Checks if this Creature is owned by the player with the given
     * ownerNumber.
     *
     * @param playerNumber The OwnerNumber of the Player where we need to know
     *                     if he owns this creature.
     *
     * @return True if the given player owns this creature.
     */
    public boolean isOwnedBy (int playerNumber)
    {
        return (this._owner == playerNumber);
    }

    /**
     * Called when a Creatures Health reaches 0.
     *
     * Base Function only hides this Creatures Tooltip.
     *
     * To be Overwritten if needed.
     */
    protected void onDeath ()
    {
        this.getTooltip().hide(this.getTooltip());
        AnimationController.getInstance().playAnimationOnTile(Animations.DEATH,
                                                                this.getTile());
```

```
        MapController.getInstance().removeObjectFromMap(this.getTile());
    }

    /**
     * Called when this Creature performs a Ranged Attack.
     *
     * Base Function only relevant for Creatures that have the Attribute "One
     * Time Shot". Will set this Creatures Attack Range to 0 once it has
     * performed its One Time Ranged Attack.
     *
     * To be Overwritten if needed.
     */
    public void onRangedAttack ()
    {
        if (this.hasAttribute(Attribute.OneTimeShot))
          {
            this._attackRange = GameObjectConstants
                                                .DEFAULT_ONETIMESHOT_ATTACK_RANGE;
            this._rangedAttack = GameObjectConstants
                                              .DEFAULT_ONETIMESHOT_RANGED_ATTACK;
          }
    }

    /**
     * To be overwritten. All Overwrites must call super() first to ensure that
     * the hitanimation is always played onhit. All effects defines for this
     * method will be executed when this creature takes any damage.
     */
    protected void onHit ()
    {
        AnimationController.getInstance().playAnimationOnTile(Animations.HIT,
                                                                  this.getTile());
    }

    /**
     * Returns this Creature as a String of its relevant Values.
     *
     * @return This Creatures as a String.
     */
    @Override
    public String toString ()
    {
        StringBuilder sb = new StringBuilder();

        sb.append(super.toString());
        sb.append("Owner: ").append(this._owner).append('\n');
        sb.append("Attack: ").append(this._attack).append('\n');
        sb.append("Health: ").append(this._health).append('\n');
        sb.append("Movement Radius: ").append(this._movementpoints).append('\n');

        return sb.toString();
    }

    /**
     * reduces the health of the CreatureAdapter if damage is insufficient to
     * kill. then returns -1. otherwise - if the CreatureAdapter is killed -
     * returns damage dealt but does not reduce health (because it'll be removed
     * from the map anyway)
     *
     * Inherited from IAttackable
     *
     * @param damage desired damage value of this creature.
     */
    @Override
    public int reduceHealth (int damage)
    {
```

```
        if (this._health > damage)
          {
            this._health -= damage;
            this.onHit();
            return -1;
          }
        else if (this._health == damage)
          {
            this.onDeath();
            return damage;
          }
        else
          {
            this.onDeath();
            return (damage - (damage - this._health));
          }

    }

    /**
     * restores the health of this Creature.
     *
     * Inherited from IAttackable
     *
     * @param restore the amount of health to be restored.
     */
    @Override
    public int restoreHealth (int restore)
    {
        if (this._health < this._maxHealth)
          {
            if (this._maxHealth - this._health < restore)
              {
                restore = this._maxHealth - this._health;
                this._health = this._maxHealth;
              }
            else
              {
                this._health += restore;
              }
          }
        else
          {
            restore = 0;
          }
        return restore;
    }

}
```

6.21.3 GameObject

Listing 79: GameObject.java

```
package Model.GameObject;

import Model.Map.Tile;
import static Resource.Constants.GeneralConstants.GameConstants.TOOLTIP_DEFAULT_NAME;
import static Resource.Constants.GeneralConstants.GameObjectConstants.SIZE;
import static Resource.Constants.GeneralConstants.NetworkConstants.EMPTY;
import Resource.Images.Images;
import View.Tooltip.GOTooltip;
import View.Tooltip.Tooltip;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.logging.Level;
```

```
13 import java.util.logging.Logger;
14 import javafx.scene.image.Image;

16 /**
17  * The class GameObject represents the Objects that should reserve a Tile on the
18  * Map.
19  */
20 public abstract class GameObject
21 {

23     /**
24      * The representing Image of this GameObject.
25      */
26     private Image _image = null;

28     /**
29      * The size of this GameObject on the Map.
30      * Relevant for finding out whether Ranged Attacks can pass through (over)
31      * this Object or whether flying Creatures can fly over this Object.
32      */
33     private int _size = 0;

35     /**
36      * The Name of this GameObject.
37      */
38     private String _name = null;

40     /**
41      * The Tile on the Map this GameObject currently occupies.
42      */
43     private Tile _tile = null;

45     /**
46      * This GameObjects Tooltip.
47      * Default Tooltip only shows this GameObjects Name.
48      */
49     private Tooltip _tooltip = new GOTooltip(TOOLTIP_DEFAULT_NAME);

51     /**
52      * Getter for this GameObjects Tooltip.
53      *
54      * @return This GameObjects Tooltip.
55      */
56     public Tooltip getTooltip ()
57     {
58         return this._tooltip;
59     }

61     /**
62      * Setter for this GameObjects Tooltip.
63      *
64      * @param tip This GameObjects Tooltip will be set to the given Tooltip.
65      */
66     protected void setTooltip (Tooltip tip)
67     {
68         this._tooltip = tip;
69     }

71     /**
72      * Getter for the Name of this GameObject.
73      *
74      * @return This GameObjects Name.
75      */
76     public String getName ()
77     {
78         return this._name;
```

```
    }

    /**
     * Setter for the Name of this GameObject.
     *
     * @param name This GameObjects Name will be set to the given String.
     */
    protected void setName (String name)
    {
        this._name = name;
        this._tooltip.setTitle(name);
    }

    /**
     * Getter for the Size Value of this GameObject.
     *
     * @return This GameObjects Size on the Map.
     */
    public int getSize ()
    {
        return this._size;
    }

    /**
     * Setter for the Size Value of this GameObject.
     *
     * @param size This GameObjects Size Value will be set to the given Value.
     */
    public void setSize (int size)
    {
        this._size = size;
        this._tooltip.addInfo(SIZE, EMPTY + size);
    }

    /**
     * Getter for the Tile this GameObject currently occupies.
     *
     * @return The Tile this GameObject thinks it is currently occupying
     */
    public Tile getTile ()
    {
        return this._tile;
    }

    /**
     * Setter for the Tile this GameObject will be occupying.
     *
     * @param tile This GameObject will think that it occupies the given Tile.
     */
    public void setTile (Tile tile)
    {
        this._tile = tile;
    }

    /**
     * Getter for this GameObjects representing Image.
     *
     * @return This GameObjects Image.
     */
    public Image getImage ()
    {
        if (this._image != null)
          {
            return this._image;
          }
        try
```

```
            {
              return new Image(new FileInputStream(Images.NO_IMAGE));
            }
          catch (FileNotFoundException ex)
            {
              Logger.getLogger(GameObject.class.getName())
                                                    .log(Level.SEVERE, null, ex);
            }
          return null;
      }

      /**
       * Setter for this GameObjects representing Image.
       *
       * Will try to set this GameObjects Image to the Image it finds on the given
       * Path.
       * If possible, only use the Paths which are stored in our Image Interfaces.
       *
       * @param path The Path to the Image File.
       */
      public void setImage (String path)
      {
          try
            {
              this._image = new Image(new FileInputStream(path));
            }
          catch (FileNotFoundException ex)
            {
              Logger.getLogger(GameObject.class.getName())
                                                    .log(Level.SEVERE, null, ex);
            }
      }

      /**
       * Transforms this GameObject to a String that can be used to display
       * relevant Information about this GameObject.
       *
       * @return String This GameObject as a String.
       */
      @Override
      public String toString ()
      {
          StringBuilder sb = new StringBuilder();

          sb.append("Name: ").append(this._name).append('\n');
          sb.append("Size: ").append(this._size).append('\n');
          sb.append("Axialtransform: ").append(this._tile.toString()).append('\n');

          return sb.toString();
      }

}
```

6.21.4 GameObjects

Listing 80: GameObjects.java

```
package Model.GameObject;

/**
 * This Enumeration is a list of all types of GameObjects that currently exist
 * within this Game.
 */
public enum GameObjects
{
```

```
    /**
     * Type of GameObject that defines it as not being of any of the Types
     * defined in this List.
     */
    noGO,

    /**
     * GameObjects of this Type Count as Obstacles.
     */
    Obstacle,

    /**
     * GameObjects of this Type Count as Mana Crystals.
     */
    ManaCrystal,

    /**
     * GameObjects of this Type Count as a Creature.
     */
    CreatureAdapter,

    /**
     * GameObjects of this Type Count as a Creature that has been summoned by
     * one of the Players.
     */
    SummonableMinion,

    /**
     * GameObjects of this Type Count as a Player Character.
     */
    PlayerCharacter,

    /**
     * GameObjects of this Type Count as a Object that has not been defined.
     */
    nonDefinedObject
}
```

6.21.5 IAttackable

Listing 81: IAttackable.java

```
package Model.GameObject;

/**
 * This Interface is to be Implemented by any GameObject that can be Attacked.
 * (Not only Creatures)
 */
public interface IAttackable
{

    /**
     * Getter for this Objects Physical Resistance Value.
     *
     * @return This Objects Physical Resistance Value.
     */
    public int _getPhysRes ();

    /**
     * Getter for this Objects Magical Resistance Value.
     *
     * @return This Objects Magical Resistance Value.
     */
    public int _getMagicRes ();

    /**
```

```
     * Reduces the Health of this Object.
     *
     * @param damage desired damage value of this Object.
     *
     * @return the amount of damage actually dealt, which can differ from the
     *         input if for example the damage is greater than remaining health
     *         and the creature dies. Returns -1 if the amount of damage given
     *         in was dealt and the creature doesn't die.
     *
     */
    public int reduceHealth (int damage);

    /**
     * Restores the Health of this Object.
     *
     * @param restore the amount of health to be restored.
     *
     * @return the amount of health that has actually been restored, which can
     *         differ from the input if for example the input restore is greater
     *         than missing health
     */
    public int restoreHealth (int restore);

}
```

6.21.6 ManaCrystal

Listing 82: ManaCrystal.java

```
package Model.GameObject;

import static Resource.Constants.GeneralConstants.GameObjectConstants
              .*;
import Resource.Images.
        Images;
import View.Tooltip.
        GOTooltip;

/**
 * The class ManaCrystal represents values of Mana on the tiles on the map that
 * could be collected by the players to fill there Mana.
 */
public class ManaCrystal extends GameObject
{

    /**
     * The Amount of Mana that is gained by a Player when his Player Character
     * walks over this Mana Crystal.
     */
    private int _manaGain = 0;

    /**
     * Default Constructor.
     * Initialises this Mana Crystals Image and Tooltip.
     */
    public ManaCrystal ()
    {
        this._manaGain = DEFAULT_MANA_CRYSTAL_GAIN;
        this.setImage(Images.MANA_CRYSTAL);
        this.setTooltip(new GOTooltip(MANA_CRYSTAL_TOOLTIP));
        this.getTooltip().addInfo(MANA_CRYSTAL_TOOLTIP_GRANTS, this._manaGain
                + MANA_CRYSTAL_TOOLTIP_MANA);
    }

    /**
     * Getter for this Mana Crystals Mana Gain Value.
```

```
     *
     * @return This Mana Crystals Mana Gain Value
     */
    public int getManaGain ()
    {
        return this._manaGain;
    }

    /**
     * Setter for this Mana Crystals Mana Gain Value.
     *
     * @param manaAmount This Mana Crystals Gain Value will be set to the given
     *                   Value.
     */
    public void setManaGain (int manaAmount)
    {
        this._manaGain = manaAmount;
    }

    /**
     * Transforms this ManaCrystal into a String that can be used to display
     * relevant Information about it.
     *
     * @return - This Mana Crystal as a String.
     */
    @Override
    public String toString ()
    {
        StringBuilder sb = new StringBuilder();

        sb.append(super.toString());
        sb.append("Mana amount: ").append(this._manaGain).append('\n');

        return sb.toString();
    }

}
```

6.21.7 Obstacle

Listing 83: Obstacle.java

```
package Model.GameObject;

import static Resource.Constants.GeneralConstants.GameObjectConstants
                                                .DEFAULT_OBSTACLE_SIZE;
import static Resource.Constants.GeneralConstants.GameObjectConstants
                                                        .OBSTACLE_NAME;
import Resource.Images.Images;

/**
 * The class Obstacle represents all the Tiles that shouldn't be entered by
 * PlayerCharacters or Summonable Minions.
 */
public class Obstacle extends GameObject
{

    /**
     * Default Constructor.
     * Sets this Obstacles Name, Size and Image.
     */
    public Obstacle ()
    {
        this.setName(OBSTACLE_NAME);
        this.setSize(DEFAULT_OBSTACLE_SIZE);
        this.setImage(Images.OBSTACLE);
```

```
    }
}
```

6.21.8 PlayerCharacter

Listing 84: PlayerCharacter.java

```
package Model.GameObject;

import Resource.Constants.MinionConstants.PlayerCharacterMinionConstants;
import Resource.Images.Images;
import View.PopUps.WinPopUp;
import View.Tooltip.MinionTooltip;
import java.io.Serializable;

/**
 * The PlayerCharacter class is the class for Player itself, on the map
 * represented with the wizard.
 */
public class PlayerCharacter extends CreatureAdapter implements Serializable
{

    /**
     * Constructs a new Player Character, Setting all its Default Values.
     *
     * @param name  This PlayerCharacters Name will be set to the given String.
     * @param owner This PlayerCharacters OnwerID will be set to the given
     *              Value.
     */
    public PlayerCharacter (String name, int owner)
    {
        this.initialize(name, owner);
    }

    /**
     * initializes the PlayerCharacter with all its default values.
     *
     * @param name  - name of the PlayerCharacter
     * @param owner - ownerNumber of the Player
     */
    private void initialize (String name, int owner)
    {
        this.setOwnerNumber(owner);
        this.setAttack(PlayerCharacterMinionConstants
                                        .DEFAULT_PLAYER_CHARACTER_ATTACK_VALUE);
        this.setMagicRes(PlayerCharacterMinionConstants
                                      .DEFAULT_PLAYER_CHARACTER_MAGIC_RES_VALUE);
        this.setMaxMvp(PlayerCharacterMinionConstants
                              .DEFAULT_PLAYER_CHARACTER_MAX_MOVEMENTPOINTS_VALUE);
        this.setMaxHealth(PlayerCharacterMinionConstants
                                     .DEFAULT_PLAYER_CHARACTER_MAX_HEALTH_VALUE);
        this.setPhysRes(PlayerCharacterMinionConstants
                                       .DEFAULT_PLAYER_CHARACTER_PHYS_RES_VALUE);
        this.setSize(PlayerCharacterMinionConstants
                                          .DEFAULT_PLAYER_CHARACTER_SIZE_VALUE);
        this.setName(PlayerCharacterMinionConstants
                                                .DEFAULT_PLAYER_CHARACTER_NAME);
        if (this.getOwner() == 0)
          {
            this.setImage(Images.PLAYER0_CHARACTER);
          }
        else if (this.getOwner() == 1)
          {
            this.setImage(Images.PLAYER1_CHARACTER);
          }
        this.setMovementpoints(PlayerCharacterMinionConstants
```

```
                                .DEFAULT_PLAYER_CHARACTER_MOVEMENTPOINTS_VALUE);
        this.setHealth(PlayerCharacterMinionConstants
                                        .DEFAULT_PLAYER_CHARACTER_HEALTH_VALUE);
        this.setTooltip(new MinionTooltip(this.getName(), this));
    }

    /**
     * Called when This PlayerCharacter dies.
     *
     * Currently Announced the Owner of the opposing PlayerCharacter as a Winner
     * and restarts the Game.
     */
    @Override
    protected void onDeath ()
    {
        int winner;

        if (this.getOwner() == 1)
          {
            winner = 0;
          }
        else
          {
            winner = 1;
          }

        new WinPopUp(winner).display();
    }

}
```

6.21.9 SummonableMinion

Listing 85: SummonableMinion.java

```
package Model.GameObject;

import Control.Actions.ManaPickupAction;
import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Control.Controllers.MapController;
import Control.Controllers.SoundController;
import Model.Map.Tile;
import Model.Player;
import Resource.Images.Animations;
import View.Tooltip.MinionTooltip;

/**
 * The SummonableMinion class is the super class for all Minions that can
 * be summoned by the Players within the game.
 */
public class SummonableMinion extends CreatureAdapter
{

    /**
     * Summons this Minion on a target Tile.
     *
     * @param target The Minion will be summoned on the given Tile.
     *
     * @return True if this Minion was successfully summoned.
     */
    public boolean summon (Tile target)
    {

        //Checks if the this Minion can be summoned on the target Tile.
        if (target.getGameObject() == null || ManaCrystal.class.isInstance(
```

```
                                                        target.getGameObject()))
            {

              Player currentPlayer = GameController.getInstance()
                                  .getCurrentTurnController().getCurrentPlayer();
              //Checks if the target Tile containts a Mana Crystal.
              if (ManaCrystal.class.isInstance(target.getGameObject()))
                {
                  ManaCrystal crystal = (ManaCrystal) target.getGameObject();
                  //Restores Mana to the current Player.
                  currentPlayer.restoreMana(crystal.getManaGain());
                  ManaPickupAction action = new ManaPickupAction(currentPlayer,
                                                                    crystal);
                  action.execute();
                }

              //Places The Minion on the Map.
              MapController.getInstance().placeObjectOnMap(this, target);
              currentPlayer.addCreature(this);

              //Announces that the Minion has been summoned.
              //System.out.println(this.getName() + " has been summoned on tile:
              //" + target);
              //Executes any OnSummon-Effects if existing.
              this.onSummon();

              return true;
            }

          //Announced that this Minion could not be summoned.
//          System.out.println(this.getName() + " could not be summoned.");
          return false;
      }

      /**
       * Called when this Minion is Summoned.
       *
       * To be Overwritten if needed.
       */
      protected void onSummon ()
      {
          SoundController.getInstance().getMusic().playSummonSound();
          //Playes The animation for summoning
          AnimationController.getInstance().playAnimationOnTile(Animations.SUMMON,
                                                                  this.getTile());
      }

      /**
       * Executed when a Instance of this Minion is created.
       *
       * Sets its Default Health Value and Tooltip.
       */
      @Override
      protected void init ()
      {
          this.setHealth(this.getMaxHealth());
          this.setTooltip(new MinionTooltip(this.getName(), this));
      }
}
```

6.22 Model.GameObject.Minions.Chaos

6.22.1 HellhoundMinion

Listing 86: HellhoundMinion.java

```
package Model.GameObject.Minions.Chaos;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * HellhoundMinion is a Chaos Minion that can be summoned by the Player with the Chaos Wand.
 */
public class HellhoundMinion extends SummonableMinion
{

    /**
     * Default Constructor.
     */
    public HellhoundMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Chaos.
                HellhoundMinionConstants.HELLHOUND_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Chaos.
                HellhoundMinionConstants.HELLHOUND_MINION_NAME);
        this.setSize(Resource.Constants.MinionConstants.Chaos
                .HellhoundMinionConstants.HELLHOUND_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Chaos.
                HellhoundMinionConstants.HELLHOUND_MINION_ATTACK);
        this.setMaxHealth(Resource.Constants.MinionConstants.Chaos.
                HellhoundMinionConstants.HELLHOUND_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Chaos.
                HellhoundMinionConstants.HELLHOUND_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Chaos.
                HellhoundMinionConstants.HELLHOUND_MINION_MAGIC_RES);
        this.setAttackRange(Resource.Constants.MinionConstants.Chaos.
                HellhoundMinionConstants.HELLHOUND_MINION_ATTACK_RANGE);
        this.setRangedAttack(Resource.Constants.MinionConstants.Chaos
                .HellhoundMinionConstants.HELLHOUND_MINION_RANGED_ATTACK);
        this.addAttribute(Attribute.Paralysing);
        this.addAttribute(Attribute.Undead);
        this.setImage(Images.HELLHOUND);
        super.init();
    }

}
```

6.22.2 HydraMinion

Listing 87: HydraMinion.java

```
package Model.GameObject.Minions.Chaos;

import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * HydraMinion is a Chaos Minion that can be summoned by the Player with the Chaos Wand.
 */
public class HydraMinion extends SummonableMinion
{

    /**
     * Default Constructor.
     */
    public HydraMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.
```

```
                Chaos.HydraMinionConstants.HYDRA_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.
                Chaos.HydraMinionConstants.HYDRA_MINION_NAME);
        this.setSize(Resource.Constants.MinionConstants.
                Chaos.HydraMinionConstants.HYDRA_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants
                .Chaos.HydraMinionConstants.HYDRA_MINION_ATTACK);
        this.setMaxHealth(Resource.Constants.MinionConstants.
                Chaos.HydraMinionConstants.HYDRA_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.
                Chaos.HydraMinionConstants.HYDRA_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.
                Chaos.HydraMinionConstants.HYDRA_MINION_MAGIC_RES);
        this.setImage(Images.HYDRA);
        super.init();
    }

}
```

6.22.3 ManticoreMinion

Listing 88: ManticoreMinion.java

```
package Model.GameObject.Minions.Chaos;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * ManticoreMinion is a Chaos Minion that can be summoned by the Player with the Chaos Wand.
 */
public class ManticoreMinion extends SummonableMinion
{
    /**
     * Default Constructor.
     */
    public ManticoreMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Chaos.
                ManticoreMinionConstants.MANTICORE_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Chaos.
                ManticoreMinionConstants.MANTICORE_NAME);
        this.setSize(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_ATTACK);
        this.setAttackRange(Resource.Constants.MinionConstants.
                Chaos.ManticoreMinionConstants.MANTICORE_MINION_ATTACK_RANGE);
        this.setRangedAttack(Resource.Constants.MinionConstants.
                Chaos.ManticoreMinionConstants.MANTICORE_MINION_RANGED_ATTACK);
        this.addAttribute(Attribute.Flying);
        this.setMaxHealth(Resource.Constants.MinionConstants.Chaos.
                ManticoreMinionConstants.MANTICORE_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_MAGIC_RES);
        this.setImage(Images.MANTICORE);
        super.init();
    }
}
```

6.22.4 SkeletonMinion

Listing 89: ManticoreMinion.java

```
package Model.GameObject.Minions.Chaos;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * ManticoreMinion is a Chaos Minion that can be summoned by the Player with the Chaos Wand.
 */
public class ManticoreMinion extends SummonableMinion
{
    /**
     * Default Constructor.
     */
    public ManticoreMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Chaos.
                ManticoreMinionConstants.MANTICORE_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Chaos.
                ManticoreMinionConstants.MANTICORE_NAME);
        this.setSize(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_ATTACK);
        this.setAttackRange(Resource.Constants.MinionConstants.
                Chaos.ManticoreMinionConstants.MANTICORE_MINION_ATTACK_RANGE);
        this.setRangedAttack(Resource.Constants.MinionConstants.
                Chaos.ManticoreMinionConstants.MANTICORE_MINION_RANGED_ATTACK);
        this.addAttribute(Attribute.Flying);
        this.setMaxHealth(Resource.Constants.MinionConstants.Chaos.
                ManticoreMinionConstants.MANTICORE_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Chaos
                .ManticoreMinionConstants.MANTICORE_MINION_MAGIC_RES);
        this.setImage(Images.MANTICORE);
        super.init();
    }
}
```

6.22.5 VampireMinion

Listing 90: VampireMinion.java

```
package Model.GameObject.Minions.Chaos;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * VampireMinion is a Chaos Minion that can be summoned by the Player with the Chaos Wand.
 */
public class VampireMinion extends SummonableMinion
{
    /**
     * Default Constructor.
     */
    public VampireMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Chaos.
                VampireMinionConstants.VAMPIRE_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Chaos.
                VampireMinionConstants.VAMPIRE_NAME);
```

```
        this.setSize(Resource.Constants.MinionConstants.Chaos.
                VampireMinionConstants.VAMPIRE_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Chaos.
                VampireMinionConstants.VAMPIRE_MINION_ATTACK);
        this.addAttribute(Attribute.Leech);
        this.addAttribute(Attribute.Undead);
        this.addAttribute(Attribute.Flying);
        this.setMaxHealth(Resource.Constants.MinionConstants.Chaos.
                VampireMinionConstants.VAMPIRE_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Chaos.
                VampireMinionConstants.VAMPIRE_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Chaos.
                VampireMinionConstants.VAMPIRE_MINION_MAGIC_RES);
        this.setImage(Images.VAMPIRE);
        super.init();
    }
}
```

6.22.6 ZombieMinion

Listing 91: ZombieMinion.java

```
package Model.GameObject.Minions.Chaos;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * ZombieMinion is a Chaos Minion that can be summoned by the Player with the Chaos Wand.
 */
public class ZombieMinion extends SummonableMinion
{

    /**
     * Default Constructor.
     */
    public ZombieMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Chaos.
                ZombieMinionConstants.ZOMBIE_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Chaos.
                ZombieMinionConstants.ZOMBIE_NAME);
        this.setSize(Resource.Constants.MinionConstants.Chaos.
                ZombieMinionConstants.ZOMBIE_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Chaos.
                ZombieMinionConstants.ZOMBIE_MINION_ATTACK);
        this.addAttribute(Attribute.Undead);
        this.setMaxHealth(Resource.Constants.MinionConstants.Chaos.
                ZombieMinionConstants.ZOMBIE_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Chaos.
                ZombieMinionConstants.ZOMBIE_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Chaos.
                ZombieMinionConstants.ZOMBIE_MINION_MAGIC_RES);
        this.setImage(Images.ZOMBIE);
        super.init();
    }

}
```

6.23 Model.GameObject.Minions.Law

6.23.1 IcarusTowerMinion

Listing 92: GameIcarusTowerMinion.java

```
package Model.GameObject.Minions.Law;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * IcarusTowerMinion is a Law Minion that can be summoned by the Player with the Law Wand.
 */
public class IcarusTowerMinion extends SummonableMinion
{

    /**
     * Default Constructor.
     */
    public IcarusTowerMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Law.
            IcarusTowerMinionConstants.ICARUS_TOWER_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Law.
                IcarusTowerMinionConstants.ICARUS_TOWER_NAME);
        this.setSize(Resource.Constants.MinionConstants.Law.
                IcarusTowerMinionConstants.ICARUS_TOWER_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Law.
                IcarusTowerMinionConstants.ICARUS_TOWER_MINION_ATTACK);
        this.setAttackRange(Resource.Constants.MinionConstants.Law
                .IcarusTowerMinionConstants.ICARUS_TOWER_MINION_ATTACK_RANGE);
        this.setRangedAttack(Resource.Constants.MinionConstants.Law
                .IcarusTowerMinionConstants.ICARUS_TOWER_MINION_RANGED_ATTACK);
        this.addAttribute(Attribute.AirInterception);
        this.addAttribute(Attribute.Structure);
        this.addAttribute(Attribute.FixedLevel);
        this.addAttribute(Attribute.UndeadSlayer);
        this.setMaxHealth(Resource.Constants.MinionConstants.Law.
                IcarusTowerMinionConstants.ICARUS_TOWER_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Law.
                IcarusTowerMinionConstants.ICARUS_TOWER_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Law.
                IcarusTowerMinionConstants.ICARUS_TOWER_MINION_MAGIC_RES);
        this.setImage(Images.ICARUS_TOWER);
        super.init();
    }

}
```

6.23.2 PaladinMinion

Listing 93: PaladinMinion.java

```
package Model.GameObject.Minions.Law;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * PaladinMinion is a Law Minion that can be summoned by the Player with the Law Wand.
 */
public class PaladinMinion extends SummonableMinion
{

    /**
     * Default Constructor.
     */
```

```
    public PaladinMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Law.
                PaladinMinionConstants.PALADIN_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Law.
                PaladinMinionConstants.PALADIN_NAME);
        this.setSize(Resource.Constants.MinionConstants.Law.
                PaladinMinionConstants.PALADIN_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Law.
                PaladinMinionConstants.PALADIN_MINION_ATTACK);
        this.addAttribute(Attribute.UndeadSlayer);
        this.setMaxHealth(Resource.Constants.MinionConstants.Law.
                PaladinMinionConstants.PALADIN_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Law.
                PaladinMinionConstants.PALADIN_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Law.
                PaladinMinionConstants.PALADIN_MINION_MAGIC_RES);
        this.setImage(Images.PALADIN);
        super.init();
    }

}
```

6.23.3 PegasusMinion

Listing 94: PegasusMinion.java

```
package Model.GameObject.Minions.Law;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * PegasusMinion is a Law Minion that can be summoned by the Player with the Law Wand.
 */
public class PegasusMinion extends SummonableMinion
{

    /**
     * Default Constructor.
     */
    public PegasusMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants
                .Law.PegasusMinionConstants.PEGASUS_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Law
                .PegasusMinionConstants.PEGASUS_NAME);
        this.setSize(Resource.Constants.MinionConstants.Law.
                PegasusMinionConstants.PEGASUS_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Law.
                PegasusMinionConstants.PEGASUS_MINION_ATTACK);
        this.addAttribute(Attribute.Flying);
        this.setMaxHealth(Resource.Constants.MinionConstants.Law.
                PegasusMinionConstants.PEGASUS_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Law.
                PegasusMinionConstants.PEGASUS_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Law.
                PegasusMinionConstants.PEGASUS_MINION_MAGIC_RES);
        this.setImage(Images.PEGASUS);
        super.init();
    }

}
```

6.23.4 SapphireDragonMinion

Listing 95: SapphireDragonMinion.java

```
package Model.GameObject.Minions.Law;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * SapphireDragonMinion is a Law Minion that can be summoned by the Player with the Law Wand.
 */
public class SapphireDragonMinion extends SummonableMinion
{

    /**
     * Default constructor.
     */
    public SapphireDragonMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Law.
        SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Law.
           SapphireDragonMinionConstants.SAPPHIRE_DRAGON_NAME);
        this.setSize(Resource.Constants.MinionConstants.Law.
                SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Law.
                SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_ATTACK);
        this.setAttackRange(Resource.Constants.MinionConstants.
         Law.SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_ATTACK_RANGE);
        this.setRangedAttack(Resource.Constants.MinionConstants.
        Law.SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_RANGED_ATTACK);
        this.addAttribute(Attribute.Flying);
        this.setMaxHealth(Resource.Constants.MinionConstants.
           Law.SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Law.
                SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Law.
                SapphireDragonMinionConstants.SAPPHIRE_DRAGON_MINION_MAGIC_RES);
        this.setImage(Images.SAPPHIRE_DRAGON);
        super.init();
    }
}
```

6.24 Model.GameObject.Minions.Neutral

6.24.1 AirElementalMinion

Listing 96: AirElementalMinion.java

```
package Model.GameObject.Minions.Neutral;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * AirElementalMinion is a Neutral Minion that can be summoned by the Player with the Neutral Wand.
 */
public class AirElementalMinion extends SummonableMinion
{

    /**
```

```
     * Default constructor.
     */
    public AirElementalMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Neutral.
                AirElementalMinionConstants.AIR_ELEMENTAL_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Neutral.
                AirElementalMinionConstants.AIR_ELEMENTAL_NAME);
        this.setSize(Resource.Constants.MinionConstants.Neutral.
                AirElementalMinionConstants.AIR_ELEMENTAL_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Neutral.
                AirElementalMinionConstants.AIR_ELEMENTAL_MINION_ATTACK);
        this.addAttribute(Attribute.Flying);
        this.addAttribute(Attribute.Knockback);
        this.addAttribute(Attribute.UndeadSlayer);
        this.setMaxHealth(Resource.Constants.MinionConstants.Neutral.
                AirElementalMinionConstants.AIR_ELEMENTAL_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Neutral.
                AirElementalMinionConstants.AIR_ELEMENTAL_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Neutral.
                AirElementalMinionConstants.AIR_ELEMENTAL_MINION_MAGIC_RES);
        this.setImage(Images.AIR_ELEMENTAL);
        super.init();
    }
}
```

6.24.2 EagleMinion

Listing 97: EagleMinion.java

```
package Model.GameObject.Minions.Neutral;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * EagleMinion is a Neutral Minion that can be summoned by the Player with the Neutral Wand.
 */
public class EagleMinion extends SummonableMinion
{

    /**
     * Default constructor.
     */
    public EagleMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Neutral
                .EagleMinionConstants.EAGLE_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Neutral.
                EagleMinionConstants.EAGLE_NAME);
        this.setSize(Resource.Constants.MinionConstants.Neutral
                .EagleMinionConstants.EAGLE_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Neutral
                .EagleMinionConstants.EAGLE_MINION_ATTACK);
        this.addAttribute(Attribute.Flying);
        this.setMaxHealth(Resource.Constants.MinionConstants.Neutral
                .EagleMinionConstants.EAGLE_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Neutral
                .EagleMinionConstants.EAGLE_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Neutral
                .EagleMinionConstants.EAGLE_MINION_MAGIC_RES);
        this.setImage(Images.EAGLE);
        super.init();
    }
```

```
}
```

6.24.3 GiantMinion

Listing 98: GiantMinion.java

```
package Model.GameObject.Minions.Neutral;

import Model.GameObject.Attribute;
import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * GiantMinion is a Neutral Minion that can be summoned by the Player with the Neutral Wand.
 */
public class GiantMinion extends SummonableMinion
{

    /**
     * Default constructor.
     */
    public GiantMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_NAME);
        this.setSize(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_ATTACK);
        this.addAttribute(Attribute.OneTimeShot);
        this.setMaxHealth(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_MAX_HEALTH);
        this.setPhysRes(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_PHYS_RES);
        this.setMagicRes(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_MAGIC_RES);
        this.setRangedAttack(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_RANGED_ATTACK);
        this.setAttackRange(Resource.Constants.MinionConstants.Neutral.
                GiantMinionConstants.GIANT_MINION_ATTACK_RANGE);
        this.setImage(Images.GIANT);
        super.init();
    }

}
```

6.24.4 LionMinion

Listing 99: LionMinion.java

```
package Model.GameObject.Minions.Neutral;

import Model.GameObject.SummonableMinion;
import Resource.Images.Images;

/**
 * LionMinion is a Neutral Minion that can be summoned by the Player with the Neutral Wand.
 */
public class LionMinion extends SummonableMinion
{

    /**
     * Default constructor.
```

```
     */
    public LionMinion ()
    {
        this.setMaxMvp(Resource.Constants.MinionConstants.Neutral
                .LionMinionConstants.LION_MINION_MAX_MOVEMENTPOINTS);
        this.setName(Resource.Constants.MinionConstants.Neutral.
                LionMinionConstants.LION_NAME);
        this.setSize(Resource.Constants.MinionConstants.Neutral.
                LionMinionConstants.LION_MINION_SIZE);
        this.setAttack(Resource.Constants.MinionConstants.Neutral.
                LionMinionConstants.LION_MINION_ATTACK);
        this.setMaxHealth(Resource.Constants.MinionConstants.Neutral.
                LionMinionConstants.LION_MINION_MAX_HEALTH);
        this.setMagicRes(Resource.Constants.MinionConstants.Neutral.
                LionMinionConstants.LION_MINION_MAGIC_RES);
        this.setPhysRes(Resource.Constants.MinionConstants.Neutral.
                LionMinionConstants.LION_MINION_PHYS_RES);
        this.setImage(Images.LION);
        super.init();
    }
}
```

6.25 Model.Map

6.25.1 AxialTransform

Listing 100: AxialTransform.java

```
package Model.Map;

import static Resource.Constants.GeneralConstants.MapConstants.*;

/**
 * The AxialTransform is a way of storing coordinates on a hexagonal grid using
 * 2 axes this is used most of the time, some Methods use CubeTransform
 * instead.
 */
public class AxialTransform
{

    /**
     * [0] = d-coord; former q [1] = h-coord; former r d stands for diagonal, h
     * for horizontal because of the way the map was displayed to the console.
     */
    private int[] _coordinates = null;

    /**
     * Creates a new Axial Transform from an array of _coordinates.
     *
     * @param coordinates - _coordinates of the transform
     */
    protected AxialTransform (int[] coordinates)
    {
        this._coordinates = new int[AMOUNT_OF_AXIAL_COORDINATES];
        if (coordinates.length == AMOUNT_OF_AXIAL_COORDINATES)
          {
            this._coordinates = coordinates;
          }
    }

    /**
     * Creates a new Axial Transform from a Cube Transform.
     *
     * @param c - cube Transform of this axial Transform.
     */
```

```
    protected AxialTransform (CubeTransform c)
    {
        this._coordinates = new int[AMOUNT_OF_AXIAL_COORDINATES];
        this._coordinates[COORDINATE_0] = c.getX();
        this._coordinates[COORDINATE_1] = c.getZ();

    }

    /**
     * Creates a new axial Transform from two separate _coordinates
     *
     * @param h - h- coordinate of the Transform.
     * @param d - d- coordinate of the Transform.
     */
    public AxialTransform (int h, int d)
    {
        this._coordinates = new int[AMOUNT_OF_AXIAL_COORDINATES];
        this._coordinates[COORDINATE_0] = d;
        this._coordinates[COORDINATE_1] = h;
    }

    /**
     * Getter for the h-coordinate.
     *
     * @return - this Transforms h-coordinate.
     */
    public int getH ()
    {
        return this._coordinates[COORDINATE_1];
    }

    /**
     * Getter for the d-coordinate.
     *
     * @return - this Transforms d-coordinate.
     */
    public int getD ()
    {
        return this._coordinates[COORDINATE_0];
    }

    /**
     * Adds two AxialTransforms.
     *
     * @param a - Transform 1.
     * @param b - Transform 2.
     *
     * @return - new Transform that has each coordinate added from both input-
     *          Transforms.
     */
    public static AxialTransform add (AxialTransform a, AxialTransform b)
    {
        return new AxialTransform(a.getH() + b.getH(), a.getD() + b.getD());
    }

    /**
     * Multiplies a Transform by a factor.
     *
     * @param t     -   Transform to multiply.
     * @param factor - Factor to multiply by.
     *
     * @return new Transform with both _coordinates multiplied by the factor.
     */
    public static AxialTransform multiply (AxialTransform t, int factor)
    {
        return new AxialTransform(t.getH() * factor, t.getD() * factor);
```

```
    }

    /**
     * Checks if this Transfrom is equal to a given Transform, two Transforms
     * are equal, if all their coordinates are equal.
     *
     * @param t - Transform to compare this one to.
     *
     * @return - true if both _coordinates are equal. otherwise return false.
     */
    public boolean equals (AxialTransform t)
    {
        return ((this.getD() == t.getD()) && (this.getH() == t.getH()));
    }

    /**
     * Produces a hash with the h and d coordinate of the specific tile.
     *
     * @return - simple hashCode to access Tiles in HashMap
     */
    @Override
    public int hashCode ()
    {
        return getH() + getD() * TO_HASH;
    }

    /**
     * The coordinates of this Tile as a String.
     *
     * @return - String Representation [h,d].
     */
    @Override
    public String toString ()
    {
        StringBuilder sb = new StringBuilder();
        sb.append(this.getH());
        sb.append(",");
        sb.append(this.getD());
        return sb.toString();
    }

}
```

6.25.2 CubeTransform

Listing 101: CubeTransform.java

```
package Model.Map;

import static Resource.Constants.GeneralConstants.MapConstants.*;

/**
 * The CubeTransform is a way of storing coordinates on a hexagonal grid using 3
 * axes, more commonly AxialTransform is used, some calculations are easier when
 * using CubeTransform, this is because for cubeTransforms x+y+z = 0 .
 */
public class CubeTransform
{

    /**
     * Coordinates of this Cube Transform, 0=x, 1=y, 2=z.
     */
    private int[] _coordinates = null;

    /**
     * Creates a new Cube Transform from an Axial Transform.
```

```
     *
     * @param a - Axial Transform to create Cube Transform from.
     */
    protected CubeTransform (AxialTransform a)
    {
        this._coordinates = new int[AMOUNT_OF_CUBE_COORDINATES];
        this._coordinates[COORDINATE_0] = a.getD();
        this._coordinates[COORDINATE_1] = -a.getD() - a.getH();
        this._coordinates[COORDINATE_2] = a.getH();
    }

    /**
     * Creates a new Cube Transform from three coordinates.
     *
     * @param X - x-coordinate of this cube Transform.
     * @param Y - y-coordinate of this cube Transform.
     * @param Z - z-coordinate of this cube Transform.
     */
    protected CubeTransform (int X, int Y, int Z)
    {
        this._coordinates = new int[AMOUNT_OF_CUBE_COORDINATES];
        this._coordinates[COORDINATE_0] = X;
        this._coordinates[COORDINATE_1] = Y;
        this._coordinates[COORDINATE_2] = Z;
    }

    /**
     * Getter for the x-coordinate of this Cubetransform.
     *
     * @return - this Transforms x-coordinate.
     */
    public int getX ()
    {
        return this._coordinates[COORDINATE_0];
    }

    /**
     * Getter for the y-coordinate of this Cubetransform.
     *
     * @return - this Transforms y-coordinate.
     */
    public int getY ()
    {
        return this._coordinates[COORDINATE_1];
    }

    /**
     * Getter for the z-coordinate of this Cubetransform.
     *
     * @return - this Transforms z-coordinate.
     */
    public int getZ ()
    {
        return this._coordinates[COORDINATE_2];
    }

    /**
     * Checks if this Transfrom is equal to a given Transform, two Transforms
     * are equal, if all their coordinates are equal.
     *
     * @param c - Transform to compare this one to.
     *
     * @return - True, if all three coordinates of both Transforms are equal.
     */
    public boolean equals (CubeTransform c)
    {
```

```
        return ((this.getX() == c.getX()) && (this.getY() == c.getY()) &&
                                                (this.getZ() == c.getZ()));
    }

    /**
     * Coordinates of this Cubetransform as a String.
     *
     * @return - String Representation [x,y,z].
     */
    @Override
    public String toString ()
    {
        StringBuilder sb = new StringBuilder();
        sb.append(this.getX() + ' ' + this.getY() + ' ' + this.getZ());
        return sb.toString();
    }

}
```

6.25.3 FloatCubeTransform

Listing 102: FloatCubeTransform.java

```
package Model.Map;

import static Resource.Constants.GeneralConstants.MapConstants.*;

/**
 * Occasionally methods produce CubeTransforms with float coordinates, namely
 * this happens in the getLine method in Map, FloatCubeTransform is used to
 * convert these to CobeTransforms or AxialTransforms.
 */
public class FloatCubeTransform
{

    /**
     * This FloatCubeTransform's coordinates, 0=x, 1=y, 2=z.
     */
    private float[] _coordinates = null;

    /**
     * Creates a new Cube Transform from three coordinates.
     *
     * @param X - x-coordinate of this cube Transform.
     * @param Y - y-coordinate of this cube Transform.
     * @param Z - z-coordinate of this cube Transform.
     */
    protected FloatCubeTransform (float X, float Y, float Z)
    {
        this._coordinates = new float[AMOUNT_OF_FLOAT_CUBE_COORDINATES];
        this._coordinates[COORDINATE_0] = X;
        this._coordinates[COORDINATE_1] = Y;
        this._coordinates[COORDINATE_2] = Z;
    }

    /**
     * Rounds all three coordinates to integer, so that a normal cube Transform
     * can be constructed, while doing this keeps the constraint x+y+z=0, that
     * defines cube coordinates for hexagonal maps.
     *
     * @return - new CubeTransfrom, with this FC-Transform's rounded coordinates.
     */
    protected CubeTransform toCubeTransform ()
    {
        //round
        int rx = Math.round(this._coordinates[COORDINATE_0]);
```

```
        int ry = Math.round(this._coordinates[COORDINATE_1]);
        int rz = Math.round(this._coordinates[COORDINATE_2]);

        //differences
        float diffX = Math.abs(rx - this._coordinates[COORDINATE_0]);
        float diffY = Math.abs(ry - this._coordinates[COORDINATE_1]);
        float diffZ = Math.abs(rz - this._coordinates[COORDINATE_2]);

        //adjust coordinate with greatest difference according to x+y+z=0.
        if ((diffX >= diffY) && (diffX >= diffZ))
          {
            rx = -ry - rz;
          }
        else if (diffY >= diffZ)
          {
            ry = -rx - rz;
          }
        else
          {
            rz = -rx - ry;
          }

        return new CubeTransform(rx, ry, rz);
    }

    /**
     * Converts this FC-Transform to an axial Transform, does so by first
     * converting to a cube Transform.
     *
     * @return - new AxialTransform, with this FC-Transform's rounded coordinates.
     */
    protected AxialTransform toAxialTransform ()
    {
        return new AxialTransform(this.toCubeTransform());
    }

}
```

6.25.4 Map

Listing 103: Map.java

```
package Model.Map;

import Model.GameObject.GameObject;
import Model.GameObject.Obstacle;
import Resource.Constants.GeneralConstants.MapConstants;
import static Resource.Constants.GeneralConstants.MapConstants.AXIAL_DIRECTIONS;
import static Resource.Constants.GeneralConstants.MapConstants.DIVIDE_BY_TWO;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * The Map keeps track of all Tiles, it also provides information about Tiles
 * and their relations, it does not change any attribbutes of Tiles, this is
 * done by the MapController.
 */
public class Map
{

    /**
     * Associates each Tile on the map with an AxialTransform's HashCode,
     * storing the map.
     */
    private HashMap<Integer, Tile> _tiles = null;
```

```
    /**
     * Constructs a new map, the needed HashMap is provided by the MapGenerator.
     *
     * @param tiles - HashMap of _tiles.
     */
    public Map (HashMap<Integer, Tile> tiles)
    {
        this._tiles = tiles;
    }

    /**
     * Returns a tile that is known by it's Transform.
     *
     * @param t - AxialTransform of which to get a Tile.
     *
     * @return - Tile associated with t
     */
    public Tile getTile (AxialTransform t)
    {
        return this._tiles.get(t.hashCode());
    }

    /**
     * Returns a Tile that is known by it's two coordinates.
     *
     * @param h - horizontal coordinate of tile.
     * @param d - diagonal coordinate of tile.
     *
     * @return - Tile associated with h and d.
     */
    public Tile getTile (int h, int d)
    {
        return this.getTile(new AxialTransform(h, d));
    }

    /**
     * Getter for the map as a Hashmap with all Tiles.
     *
     * @return HashMap of all Tiles
     */
    public HashMap<Integer, Tile> getTiles ()
    {
        return _tiles;
    }

    /**
     * Returns a list of all neighbours to a Tile, they will be ordered, if all
     * neighbours are present, as dictated by the AXIAL_DIRECTIONS in
     * MapConstants.
     *
     * @param center - Tiles of which to get all neighbours.
     *
     * @return - up to 6 Tiles adjacent center. less if on edge of the map.
     */
    public ArrayList<Tile> getNeighbours (Tile center)
    {
        ArrayList<Tile> results = new ArrayList<>();
        for (int i = 0; i < MapConstants.AXIAL_DIRECTIONS.length; i++)
          {
            if (getTile(AxialTransform.add(center.getAxialTransform(),
                                    MapConstants.AXIAL_DIRECTIONS[i])) != null)
              {
              results.add(getTile(AxialTransform.add(center.getAxialTransform(),
                          MapConstants.AXIAL_DIRECTIONS[i])));
              }
          }
```

```
        return results;
    }

    /**
     * Calculates the Distance "as the crow flies" - over all objects.
     *
     * @param a - Tile 1
     * @param b - Tile 2
     *
     * @return - int distance between Tiles; -1 if either Tile is null.
     */
    public int getDistance (Tile a, Tile b)
    {
        if ((a != null) && (b != null))
          {
            if (a.getAxialTransform().equals(b.getAxialTransform()))
              {
                return 0;
              }
            else
              {
                CubeTransform cube1 = new CubeTransform(a.getAxialTransform());
                CubeTransform cube2 = new CubeTransform(b.getAxialTransform());
                return (Math.abs(cube1.getX() - cube2.getX()) + Math.abs(cube1
                        .getY() - cube2.getY()) + Math.abs(cube1.getZ() -
                         cube2.getZ())) / DIVIDE_BY_TWO;
              }
          }
        return -1;
    }

    /**
     * Getter for a Line between two given Tiles.
     *
     * @param a - Tile 1
     * @param b - Tile 2
     *
     * @return - a list of Tiles that make up a line between the input _tiles,
     *          including them.
     */
    public ArrayList<Tile> getLine (Tile a, Tile b)
    {
        CubeTransform aCube = new CubeTransform(a.getAxialTransform());
        CubeTransform bCube = new CubeTransform(b.getAxialTransform());

        int n = getDistance(a, b);
        ArrayList<Tile> results = new ArrayList<>();
        float h = (float) (1.0 / n);      //calculation of helper outside of loop

        for (int i = 0; i < n + 1; i++)
          /*
           * n+1, because the starting tile is an
           * item of the returned ArrayList
           */
          {
            float h2 = h * i; //calculation of second helper outside of constructor
            results.add(this.getTile(new FloatCubeTransform(
                    linearInterpolation(aCube.getX(), bCube.getX(), h2),
                    linearInterpolation(aCube.getY(), bCube.getY(), h2),
                    linearInterpolation(aCube.getZ(), bCube.getZ(), h2)
            ).toAxialTransform()));
          }
        return results;
    }

    /**
```

```
     * Helper method for getLine.
     *
     * @param a - the first value
     * @param b - the second value
     * @param t - where between the two the return value is
     *
     * @return - linearInterpolation of the given values.
     */
    private float linearInterpolation (int a, int b, float t)
    {
        return (a + (b - a) * t);
    }

    /**
     * Getter for a Ring of Tiles around a given CenterTile.
     *
     * @param center - tile around which to get the Ring.
     * @param radius - Distance from the center that the ring will have.
     *
     * @return - _tiles that form a Ring around the center, with radius distance
     *          from the center tile.
     */
    public ArrayList<Tile> getRing (Tile center, int radius)
    {
        AxialTransform indexTransform = center.getAxialTransform();

        ArrayList<Tile> results = new ArrayList<>();

        indexTransform = AxialTransform.add(indexTransform,
            AxialTransform.multiply(MapConstants.AXIAL_DIRECTIONS[4], radius));

        for (int i = 0; i < AXIAL_DIRECTIONS.length; i++)
          {
            for (int j = 0; j < radius; j++)
              {
                if (this.getTile(indexTransform) != null)
                  {
                    results.add(this.getTile(indexTransform));
                  }
                indexTransform = AxialTransform.add(indexTransform,
                                                    MapConstants.AXIAL_DIRECTIONS[i]);
              }
          }
        return results;
    }

    /**
     * Multiple uses of getRing() form a "spiral"
     *
     * @param center - the centerTile
     * @param range  - the range of the requested area
     *
     * @return - all _tiles within range of the center (center tile excluded) if
     *          range smaller than 1 returns center tile.
     */
    public ArrayList<Tile> getTilesInRange (Tile center, int range)
    {
        ArrayList<Tile> results = new ArrayList<>();

        if (range <= 0)
          {
            results.add(center);
          }
        else
          {
            //loop starts at 1!
```

```
            for (int i = 1; i <= range; i++)
              {
                results.addAll(getRing(center, i));
              }
          }
        return results;
    }

    /**
     * Uses the getTilesInRange method but removes _tiles with obstacles from
     * the
     * results.
     *
     * @param center - the centerTile
     * @param range  - the Range of the requested area
     *
     * @return - _tiles in range without obstacles. Center tile if range is
     *         smaller
     *         than 1.
     */
    public ArrayList<Tile> getTilesInRangeExceptObstacles (Tile center, int range)
    {
        ArrayList<Tile> results = new ArrayList<>();

        for (Tile t : this.getTilesInRange(center, range))
          {
            if (!Obstacle.class.isInstance(t.getGameObject()))
              {
                results.add(t);
              }
          }
        return results;
    }

    /**
     * Checks lineOfSight between an object and a tile.
     *
     * @param o - Object
     * @param t - Tile
     *
     * @return - true, if the object can see the tile.
     */
    public boolean lineOfSight (GameObject o, Tile t)
    {
        return lineOfSight(o.getTile(), o.getSize(), t, 0);
    }

    /**
     * Checks lineOfSight between two objects.
     *
     * @param a - Object 1
     * @param b - Object 2
     *
     * @return - true, if the object can see the other object.
     */
    public boolean lineOfSight (GameObject a, GameObject b)
    {
        return lineOfSight(a.getTile(), a.getSize(), b.getTile(), b.getSize());
    }

    /**
     * Detremines Line of Sight.
     *
     * @param a        - tile a
     * @param oHeightA - object or creature height a
     * @param b        - tile b
```

```
     * @param oHeightB - object or creature height b
     *
     * @return - true if there is a line of sight between the given _tiles.
     */
    private boolean lineOfSight (Tile a, int oHeightA, Tile b, int oHeightB)
    {
        int heightA = a.getHeight() + oHeightA;
        int heightB = b.getHeight() + oHeightB;

        //everything can always see itself and adjacent _tiles
        if (getDistance(a, b) <= 1)
          {
            return true;
          }

        boolean loS = true;

        //check every tile on line betweeen a and b for height and obstacles.
        //this might need some work to return believable results
        for (Tile t : getLine(a, b))
          {
            if (t != null)
              {
                if ((t.getHeight() > heightA) && (t.getHeight() > heightB))
                  {
                    loS = false;
                  }
                if ((t.getGameObject() != null) && (Obstacle.class.isInstance(
                        t.getGameObject())))
                  {
                    loS = false;
                  }
              }
          }
        return loS;
    }

    /**
     * Override of toString.
     *
     * @return - the entire map as string, formatted to be displayed in the
     *          console, up to a radius of 9 doesnt work for non-rhombus shaped
     *          maps and is therefore obsolete.
     */
    @Override
    public String toString ()
    {
        StringBuilder sb = new StringBuilder();
        sb.append("mapToString not supported any more");
        return sb.toString();
    }

}
```

6.25.5 MapGenArgs

Listing 104: MapGenArgs.java

```
package Model.Map;

import static Resource.Constants.GeneralConstants.MapConstants
                                          .DEFAULT_MAP_GEN_ARGS;
import static Resource.Constants.GeneralConstants.NetworkConstants.EMPTY;

/**
 * This class exists to save settings made in the game options menu, until used
```

```
 * for loading or generating a map.
 */
public class MapGenArgs
{

    /**
     * The _shape of the map.
     */
    private static MapGenerationKeywords _shape = DEFAULT_MAP_GEN_ARGS[0];

    /**
     * The _size of the map.
     */
    private static MapGenerationKeywords _size = DEFAULT_MAP_GEN_ARGS[1];

    /**
     * The _random value of the map.
     */
    private static MapGenerationKeywords _random = DEFAULT_MAP_GEN_ARGS[2];

    /**
     * The filename of the map.
     */
    private static String _fileName = EMPTY;

    /**
     * Getter for the arguements of the map.
     *
     * @return - args
     */
    public static MapGenerationKeywords[] getArgs ()
    {
        MapGenerationKeywords[] args = new MapGenerationKeywords[3];
        args[0] = _shape;
        args[1] = _size;
        args[2] = _random;
        return args;
    }

    /**
     * Setter for the _shape of the map
     *
     * @param shape - the _shape of the map.
     */
    public static void setShape (MapGenerationKeywords shape)
    {
        MapGenArgs._shape = shape;
    }

    /**
     * Setter for the _size of the map.
     *
     * @param size - the _size the map should have.
     */
    public static void setSize (MapGenerationKeywords size)
    {
        MapGenArgs._size = size;
    }

    /**
     * Setter for the _random value of the Map.
     *
     * @param random - the _random value the map should have.
     */
    public static void setRandom (MapGenerationKeywords random)
    {
```

```
        MapGenArgs._random = random;
    }

    /**
     * Setter for the _fileName of the map.
     *
     * @param newFileName - the name of the .map file.
     */
    public static void setFileName (String newFileName)
    {
        _fileName = newFileName;
    }

    /**
     * Getter for the _fileName of the map.
     *
     * @return - the name of the .map file.
     */
    public static String getFileName()
    {
        return _fileName;
    }

}
```

6.25.6 MapGenerationKeywords

Listing 105: MapGenerationKeywords.java

```
package Model.Map;

/**
 * Arguments that can be used to generate maps with the map generator.
 *
 * @author Henning
 */
public enum MapGenerationKeywords
{
    /**
     * hexagon-shape.
     */
    hexagon,

    /**
     * rhombus-shape.
     */
    rhombus,

    /**
     * small-size.
     */
    small,

    /**
     * medium-size.
     */
    medium,

    /**
     * large-size.
     */
    large,

    /**
     * not-random. Will load a stored map.
     */
```

```
    noRandom,

    /**
     * random. will generate a new map.
     */
    random;

}
```

6.25.7 MapGenerator

Listing 106: MapGenerator.java

```
package Model.Map;

import Control.Controllers.MapController;
import Exceptions.MapTypeNotSupportedException;
import Model.GameObject.ManaCrystal;
import Model.GameObject.Obstacle;
import Resource.Constants.GeneralConstants.ControllerConstants;
import Resource.Constants.GeneralConstants.MapConstants;
import static Resource.Constants.GeneralConstants.MapConstants.*;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * The MapGenerator generates a map according to given arguments. It can also
 * save and load maps to/from .txt files.
 */
public class MapGenerator
{

    /**
     * tiles for placing the playerCharacters symmetrically.
     */
    private static Tile[] _wizardSpawnsArr = null;

    /**
     * Getter for the the Wizard spawns.
     *
     * @return - The Wizard spawns in a Tile Array.
     */
    public static Tile[] getWizardSpawns ()
    {
        return _wizardSpawnsArr;
    }

    /**
     * Parses the mapGenerationKeywords and then uses the appropriate map
     * generation method.
     *
     * @param args: an array of MapGenerationKeywords, that contains, in this
     *              order: the shape; the size; random or noRandom.
     *
     * @return HashMap of tiles, containing the generated map.
     *
     * @throws Exceptions.MapTypeNotSupportedException Thrown if this method
     *                                                 tries to create a map
     *                                                 with an unknown type.
     */
    public static HashMap<Integer, Tile> generateMap (MapGenerationKeywords[] args)
                                            throws MapTypeNotSupportedException
    {
```

```
        switch (args[2])
          {
            case random:
                switch (args[0])
                  {
                    case rhombus:
                        return generateRhombusMap(args[1]);
                    case hexagon:
                        return generateHexagonMap(args[1]);
                    default:
                        throw new MapTypeNotSupportedException();
                  }
            case noRandom:
                switch (args[0])
                  {
                    case hexagon:
                        switch (args[1])
                          {
                            case small:
                                return loadMap(MAP_NAME_SMALL);
                            case medium:
                                return loadMap(MAP_NAME_MEDIUM);
                            case large:
                                return loadMap(MAP_NAME_LARGE);
                            default:
                                throw new MapTypeNotSupportedException();
                          }
                    default:
                        throw new MapTypeNotSupportedException();
                  }
            default:
                throw new MapTypeNotSupportedException();
          }
    }

    /**
     * Generates a Rhombus-shaped map, where the edges' length are equal to the
     * DIAMETER constant in MapConstants.
     *
     * @param size Size of the map.
     *
     * @return HashMap containing the generated map.
     */
    private static HashMap<Integer, Tile> generateRhombusMap (
                                                        MapGenerationKeywords size)
    {
        HashMap<Integer, Tile> tiles = new HashMap<>();
        PerlinNoise noise = new PerlinNoise();

        float sizeModifier = getSizeModifier(size);

        for (int i = 0; i < MapConstants.DIAMETER * sizeModifier; i++)
          {
            for (int j = 0; j < MapConstants.DIAMETER * sizeModifier; j++)
              {
                AxialTransform t = new AxialTransform(i, j);
                if (tiles.get(t.hashCode()) == null)
                  {
                    tiles.put(t.hashCode(), generateTile(t, noise));
                  }
                else
                  {
                    //this technically should never happen but just in case...
                    System.err.println(TILE_OVERLAP);
                  }
              }
```

```
            }
        noise = new PerlinNoise();
        for (Tile t : tiles.values())
            {
                generateFeatures(t, noise);
            }
        setWizardSpawns(tiles);

        return tiles;
    }

    /**
     * Generates a Hexagon-shaped map, where the diameter is equal to the
     * DIAMETER constant in MapConstants.
     *
     * @param size Size of the map.
     *
     * @return HashMap containing the generated map.
     */
    private static HashMap<Integer, Tile> generateHexagonMap (
                                                    MapGenerationKeywords size)
    {
        HashMap<Integer, Tile> tiles = new HashMap<>();
        PerlinNoise noise = new PerlinNoise();

        float sizeModifier = getSizeModifier(size);
        int radius = (int) ((MapConstants.DIAMETER * sizeModifier) / 2f);

        for (int i = 0; i < radius * 2 + 1; i++)
            {
                for (int j = Math.max(radius - (i), 0); j < Math.min(radius * 2 + 1,
                                                radius * 2 + 1 - (i - radius)); j++)
                    {
                        AxialTransform t = new AxialTransform(i, j);
                        if (tiles.get(t.hashCode()) == null)
                            {
                                tiles.put(t.hashCode(), generateTile(t, noise));
                            }
                        else
                            {
                                //this technically should never happen but just in case...
                                System.err.println(TILE_OVERLAP);
                            }
                    }
            }
        noise = new PerlinNoise();
        for (Tile t : tiles.values())
            {
                generateFeatures(t, noise);
            }
        setWizardSpawns(tiles);

        return tiles;
    }

    /**
     *
     * @param size desired map size.
     *
     * @return Float factor to multiply map size by.
     */
    private static float getSizeModifier (MapGenerationKeywords size)
    {
        float sizeModifier = 0;
        switch (size)
            {
```

```
            case small:
                sizeModifier = MapConstants.SIZE_MODIFIERS[0];
                break;
            case medium:
                sizeModifier = MapConstants.SIZE_MODIFIERS[1];
                break;
            case large:
                sizeModifier = MapConstants.SIZE_MODIFIERS[2];
                break;
        }
    return sizeModifier;
}

/**
 * generates Features for a tile. Features are Obstacles and ManaCrystals.
 *
 * @param t tile to generate feature for.
 * @param n Noise to use for generation.
 */
private static void generateFeatures (Tile t, PerlinNoise n)
{
    /*
        Cannot use MapController.placeObjectOnMap because
        the MapController is not yet initialized when this method is called
    */
    if ((n.noise(t.getAxialTransform().getD(), t.getAxialTransform().getH(),
            MapConstants.FEATURE_NOISE_SCALE)) >
                                    MapConstants.DEFAULT_OBSTACLE_AMOUNT)
        {
        Obstacle o = new Obstacle();
        o.setTile(t);
        t.setGameObject(o);
        }
    if ((n.noise(t.getAxialTransform().getD(), t.getAxialTransform().getH(),
        MapConstants.FEATURE_NOISE_SCALE)) <
                                     MapConstants.DEFAULT_CRYSTAL_AMOUNT)
        {
        ManaCrystal m = new ManaCrystal();
        m.setTile(t);
        t.setGameObject(m);
        }
}

/**
 * Generates a Tile. Transform is given by the generateMap() method. Height
 * is determined by the given Noise.
 *
 * @param t Axial Transform of the tile to generate.
 * @param n Noise to generate height with.
 *
 * @return
 */
private static Tile generateTile (AxialTransform t, PerlinNoise n)
{
    Tile tile = new Tile(t, (int) Math.round((n.noise(t.getD(), t.getH(),
                            MapConstants.TERRAIN_NOISE_SCALE)) * 2.0) + 1);
    return tile;
}

/**
 * Saves the map to a .map.; each tile is represented by one line,
 * containing the following information:
 * [h-coordinate|d-coordinate|height|other] other is 0 if the tile is empty,
 * 1 if there's an obstacle on the tile, 2 if there is a manaCrystal
 *
 * @param name The Name this map is to be saved by.
```

```
     *
     * @return true if saving was successful, false if not.
     */
    public static boolean saveMap (String name)
    {
        boolean success = false;
        FileWriter out = null;

        try
          {
            out = new FileWriter(name + MAP_DATAFORMAT);

            for (Tile t : MapController.getInstance().getMap().getTiles().values())
              {
                out.write(t.saveToString());
              }
            success = true;
          }
        catch (IOException e)
          {
          }
        finally
          {
            if (out != null)
              {
                try
                  {
                    out.close();
                  }
                catch (IOException e)
                  {
                  }
              }
          }

        return success;
    }

    /**
     * Loads a Map from a .map file each tile is represented with these values
     * in that order: [h-coordinate|d-coordinate|height|other] other is 0 if the
     * tile is empty, 1 if there's an obstacle on the tile, 2 if there is a
     * manaCrystal and 3 if the tile is set as a spawn point for a wizard. If
     * more than 2 tiles in a map are set as spawns only the first 2 will be
     * used.
     *
     * @param fileName The File Name of the map that is to be loaded
     *
     * @return The Map in a HashMap format.
     */
    @SuppressWarnings(FALLTHROUGH)
    public static HashMap<Integer, Tile> loadMap (String fileName)
    {
        ArrayList<String> strings = new ArrayList<>();

        FileReader in = null;

        HashMap<Integer, Tile> tiles = new HashMap<>();

        StringBuilder fileNameString = new StringBuilder();
        fileNameString.append(fileName);
        fileNameString.append(MAP_DATAFORMAT);

        try
          {
            in = new FileReader(fileNameString.toString());
```

```
319             StringBuilder sb = new StringBuilder();
320             int c;
321             c = in.read();

323             do
324               {
325                 switch (c)
326                   {
327                     case -1: ;/*File end. Fallthrough,
328                                zur ausgabe des Letzten StringBuilders,
329                                nach Ende des In-Streams*/
330                     case 13:
331                         break;
332                     case 10: //new line
333                         String meinString = sb.toString();
334                         strings.add(meinString);
335                         sb = new StringBuilder();
336                         break;
337                     /*
338                      * case 44: //comma
339                      * String meinString = sb.toString();
340                      * int d = Integer.parseInt(meinString);
341                      * out.write(d);
342                      * sb = new StringBuilder();
343                      * ;
344                      * break;
345                      */
346                     default:
347                         sb.append((char) (c));
348                         break;
349                   }
350                 c = in.read();
351               }
352             while (c != (-1));
353           }
354         catch (IOException e)
355           {
356           }
357         finally
358           {
359             if (in != null)
360               {
361                 try
362                   {
363                     in.close();
364                   }
365                 catch (IOException e)
366                   {
367                   }
368               }
369           }

371         ArrayList<Tile> wizardSpawns = new ArrayList<>();

373         for (String s : strings)
374           {
375             String[] sep = s.split(",");

377             int h = Integer.parseInt(sep[0]);
378             int d = Integer.parseInt(sep[1]);
379             int height = Integer.parseInt(sep[2]);

381             AxialTransform t = new AxialTransform(h, d);
382             Tile tile = new Tile(t, height);
```

```
            tiles.put(t.hashCode(), tile);

            switch (Integer.parseInt(sep[3]))
              {
                case 1:
                    MapController.getInstance().placeObjectOnMap(new Obstacle(),
                                                                        tile);
                    break;
                case 2:
                    MapController.getInstance().placeObjectOnMap(new ManaCrystal()
                                                                       , tile);
                    break;
                case 3:
                    wizardSpawns.add(tile);
                    break;
                default:
                    break;
              }
          }
        if (wizardSpawns.size() >= 2)
          {
            _wizardSpawnsArr = new Tile[2];

            wizardSpawns.toArray(_wizardSpawnsArr);
          }
        else
          {
            setWizardSpawns(tiles);
          }

        return tiles;
    }

    /**
     * Sets wizardSpawns according to ControllerConstants.
     *
     * @param map the map for which the wizard spawns have to be set.
     */
    private static void setWizardSpawns (HashMap<Integer, Tile> map)
    {
        _wizardSpawnsArr = new Tile[2];

        int minD = Integer.MAX_VALUE;
        int minH = Integer.MAX_VALUE;
        for (Tile t : map.values())
          {
            if (t.getAxialTransform().getH() < minH)
              {
                minH = t.getAxialTransform().getH();
              }

          }
        for (Tile t : map.values())
          {

            if ((t.getAxialTransform().getD() < minD) && (map.get(
                    new AxialTransform(minH, t.getAxialTransform().getD())
                                                        .hashCode()) != null))
              {
                minD = t.getAxialTransform().getD();
              }
          }

        _wizardSpawnsArr[0] = map.get(new AxialTransform(minH +
                    ControllerConstants.SPAWN_DISTANCE_FROM_EDGE, minD +
                      ControllerConstants.SPAWN_DISTANCE_FROM_EDGE).hashCode());
```

```
        int maxH = Integer.MIN_VALUE;
        int maxD = Integer.MIN_VALUE;
        for (Tile t : map.values())
          {
            if (t.getAxialTransform().getH() > maxH)
              {
                maxH = t.getAxialTransform().getH();
              }

          }
        for (Tile t : map.values())
          {

            if ((t.getAxialTransform().getD() > maxD) && (map.get(
                    new AxialTransform(maxH, t.getAxialTransform().getD())
                                                                .hashCode())) != null)
              {
                maxD = t.getAxialTransform().getD();
              }
          }

        _wizardSpawnsArr[1] = map.get(new AxialTransform(maxH -
                    ControllerConstants.SPAWN_DISTANCE_FROM_EDGE, maxD -
                      ControllerConstants.SPAWN_DISTANCE_FROM_EDGE).hashCode());

    }

}
```

6.25.8 Pathfinding

Listing 107: Pathfinding.java

```
package Model.Map;

import Control.Controllers.MapController;
import Model.GameObject.Attribute;
import Model.GameObject.CreatureAdapter;
import Model.GameObject.Obstacle;
import static Resource.Constants.GeneralConstants.MapConstants.DIVIDE_BY_TWO;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.PriorityQueue;

/**
 * The Pathfinder calculates the Path from a given start Tile to a given end
 * tile and also the movementpoints that you have to use to get there.
 */
public class Pathfinding
{

    /**
     * Instance for making the pathfinder a singleton.
     */
    private static Pathfinding _instance = new Pathfinding();

    /**
     * The Start Tile
     */
    private Tile _start = null;

    /**
     * The End Tile
     */
    private Tile _end = null;
```

```
    /**
     * A Map with the Path from the Start to the End Tile
     */
    private HashMap<Integer, Tile> _cameFrom = null;

    /**
     * A Map where to every Tile of the _path the required costs are mapped to
     */
    private HashMap<Integer, Integer> _costSoFar = null;

    /**
     * output _path, ordered from _start to _end.
     */
    private ArrayList<Tile> _path = null;

    /**
     * Calculates the movementPoints from given _start and _end tiles on a given
     * map
     *
     * @param start      Tile on which the _path starts
     * @param end        Desired _end tile
     * @param outputPath if a cohesive output _path should be logged to the
     *                   console.
     * @param ca         Creature that attempts to find a _path. Is important
     *                   for
     *                   flying, for example.
     *
     * @return neededMovementPoints
     */
    public int calcMovementPoints (Tile start, Tile end, boolean outputPath,
                                                          CreatureAdapter ca)
    {
        this._start = start;
        this._end = end;
        if (Obstacle.class.isInstance(this._end.getGameObject()) ||
                    CreatureAdapter.class.isInstance(this._end.getGameObject()))
          {
            return Integer.MAX_VALUE;
          }
        this.aStarSearch(outputPath, ca);
        return this.getHighestCost();
    }

    /**
     * Finds the Path from the current Start to the current End Tile on the
     * current Map
     *
     * @param ca         Creature that attempts to find a _path. Is important
     *                   for
     *                   flying, for example.
     * @param outputPath if a cohesive output _path should be logged to the
     *                   console.
     */
    public void aStarSearch (boolean outputPath, CreatureAdapter ca)
    {
        boolean pathFound = false;
        _cameFrom = new HashMap<>();
        _costSoFar = new HashMap<>();

        PriorityQueue<Tile> frontier = new PriorityQueue<Tile>();

        this._start.setPriority(0);
        frontier.add(this._start);
        this._cameFrom.put(this._start.hashCode(), this._start);
        this._costSoFar.put(this._start.hashCode(), 0);
```

```
        while (frontier.size() > 0)
          {
            Tile current = frontier.poll();

            if (current.hashCode() == this._end.hashCode())
              {
                pathFound = true;
                break;
              }

            for (Tile next : MapController.getInstance().getMap()
                                                        .getNeighbours(current))
              {
                if ((((CreatureAdapter.class.isInstance(current.getGameObject()))
                                        && !ca.hasAttribute(Attribute.Flying)) ||
                        (Obstacle.class.isInstance(current.getGameObject()))) &&
                        (current.hashCode() != _start.hashCode()))
                  {
                    continue;
                  }

                int heightDifference = Math.max(next.getHeight(),
                                current.getHeight()) - Math.min(next.getHeight(),
                                                        current.getHeight());

                if (((heightDifference > 1) && !(ca.hasAttribute(
                                Attribute.Flying))) || (heightDifference >= 1 &&
                                        ca.hasAttribute(Attribute.FixedLevel)))
                  {
                    continue;
                  }

                int newCost = this._costSoFar.get(current.hashCode()) +
                                                next.getMovementPointCost();

                if (!this._costSoFar.containsKey(next.hashCode()) || newCost <
                                        this._costSoFar.get(next.hashCode()))
                  {
                    this._costSoFar.put(next.hashCode(), newCost);
                    next.setPriority(newCost + this.heuristic(
                        next.getAxialTransform(), this._end.getAxialTransform()));
                    frontier.add(next);
                    this._cameFrom.put(next.hashCode(), current);

                  }
              }
          }
        if (pathFound)
          {
            generateOutputPath(outputPath);
          }
        else
          {
            this._costSoFar.put(this._start.hashCode(), Integer.MAX_VALUE);
          }
    }

    /**
     * generates from _cameFrom the total _path
     *
     * @param outputPath True if this path should be output to the console.
     */
    private void generateOutputPath (boolean outputPath)
    {
        ArrayList<Tile> pathBack = new ArrayList<>();
```

```
        Tile current = _end;

        //Path from _end to _start
        pathBack.add(current);

        //pulls tiles from the _cameFrom hash-map and puts them in reverse order.
        while (current.hashCode() != _start.hashCode())
          {
            pathBack.add(this._cameFrom.get(current.hashCode()));
            current = this._cameFrom.get(current.hashCode());
          }

        Tile[] pathBackArr = new Tile[pathBack.size()];
        pathBack.toArray(pathBackArr);

        //reverse _path; so that it now starts at the _start
        ArrayList<Tile> pathFor = new ArrayList<>();

        for (int i = pathBackArr.length - 1; i >= 0; i--)
          {
            pathFor.add(pathBackArr[i]);
          }

        this._path = pathFor;

        if (outputPath)
          {

          }
    }

    /**
     * calculates a heuristic for the aStarSearch
     *
     * @param axialTransformA
     * @param axialTransformB
     *
     * @return
     */
    private int heuristic (AxialTransform axialTransformA,
                                                AxialTransform axialTransformB)
    {
        return ((Math.abs(axialTransformA.getD() - axialTransformB.getD()) +
                Math.abs(axialTransformA.getH() - axialTransformB.getH()) +
                Math.abs(axialTransformA.getD() + axialTransformA.getH() -
                        axialTransformB.getD() - axialTransformB.getH())) /
                        DIVIDE_BY_TWO);
    }

    /**
     * finds the highest value in the _costSoFar map
     *
     * @return highest value in the _costSoFar map.
     */
    private int getHighestCost ()
    {
        int max = 0;
        for (int value : _costSoFar.values())
          {
            if (value > max)
              {
                max = value;
              }
          }
        return max;
    }
```

```
    //------------------
    //Getter and Setter
    //-----------------
    /**
     *
     * @return Instance of the Pathfinding.
     */
    public static Pathfinding getInstance ()
    {
        return _instance;
    }

    /**
     * Getter
     *
     * @return the _start tile
     */
    public Tile getStartTile ()
    {
        return this._start;
    }

    /**
     *
     * @return the generated _path.
     */
    public ArrayList<Tile> getPath ()
    {
        return this._path;
    }

    /**
     * Getter
     *
     * @return the _end tile
     */
    public Tile getEndTile ()
    {
        return this._end;
    }

    /**
     * Setter
     *
     * @param start tile to _start on
     */
    public void setStartTile (Tile start)
    {
        this._start = start;
    }

    /**
     * Setter
     *
     * @param end desired _end tile
     */
    public void setEndTile (Tile end)
    {
        this._end = end;
    }

    /**
     * Getter
     *
     * @return _cameFrom
```

```
     */
    public HashMap<Integer, Tile> getCameFrom ()
    {
        return this._cameFrom;
    }

    /**
     * Getter
     *
     * @return _costSoFar
     */
    public HashMap<Integer, Integer> getCostSoFar ()
    {
        return this._costSoFar;
    }

}
```

6.25.9 PerlinNoise

Listing 108: PerlinNoise.java

```
package Model.Map;

import static Resource.Constants.GeneralConstants.MapConstants.SHORT_DEFAULT;
import java.util.Random;

/**
 * Perlin noise is a tool to generating terrain that is random but adjacent
 * areas depend on one another so that cohesive terrain is the result. If
 * instead purely random numbers were used, the map would be too rough and
 * likely impossible to navigate for non-flying creatures.
 */
public class PerlinNoise
{

    /**
     * Creates a new Perlin Noise.
     */
    public PerlinNoise ()
    {
        //fill p with random ints, that fit in the Short value range
        Random random = new Random();

        for (int i = 0; i < _p.length; i++)
          {
            _p[i] = (short) (random.nextInt(Short.MAX_VALUE + 1));
          }
    }

    /**
     *
     */
    private short _p[] = new short[SHORT_DEFAULT];

    /**
     * Math class only provides floor method that returns double. this doesn't
     * need to be cast into int afterwards and according to some guy on
     * StackOverflow is faster, which i don't necessarily believe but not
     * needing to cast is still convenient.
     *
     * @param x
     *
     * @return largest int, less then or equal to x
     */
    private int floor (double x)
```

```
    {
        return x >= 0 ? (int) x : (int) x - 1;
    }

    private double linearInterpolation (double a, double b, double t)
    {
        return (a + (b - a) * t);
    }

    /**
     *
     * @param x     x-coordinate to get noise value from.
     * @param y     y-coordinate to get noise value from.
     * @param scale factor by which to multiply the coordinates. basically the
     *              "roughness". the smaller scale, the more flat the noise
     *              appears.
     *
     * @return this noises value at the specified x and y position for the given
     *         scale.
     */
    public double noise (double x, double y, float scale)
    {
        x *= scale;
        y *= scale;

        //X & Y are the coordinates of the cell that contains the point (x,y)
        int X = floor(x);
        int Y = floor(y);

        //repurpose x,y to now be the relative location of the point in cell
        x = x - X;
        y = y - Y;

        /*getting values for four corners of current cell by "hashing"
        x and y to get access the same index of p for the same x and y. */
        int x0y0 = _p[3 * X + 5 * Y & 255];
        /*
            System.out.println("accessing p at " +
            (3 * X + 5*Y & 255) + " Value: " + x0y0);
        */
        int x0y1 = _p[3 * X + 5 * Y + 1 & 255];
        /*
            System.out.println("accessing p at " +
            3 * X + 5*Y + 1 & 255) + " Value: " + x0y1);*
        */
        int x1y0 = _p[3 * (X + 1) + 5 * Y & 255];
        /*
            System.out.println("accessing p at " +
            (3 * (X + 1) + 5*Y & 255) + " Value: " + x1y0);
        */
        int x1y1 = _p[3 * (X + 1) + 5 * Y + 1 & 255];
        /*
            System.out.println("accessing p at " +
            (3 * (X + 1) + 5*Y + 1 & 255) + " Value: " + x1y1);
        */

        double interpol1 = linearInterpolation(x0y0, x1y0, x);
        double interpol2 = linearInterpolation(x0y1, x1y1, y);

        double finalInterpol = linearInterpolation(interpol1, interpol2, y);

        return (finalInterpol / Short.MAX_VALUE);
    }

}
```

6.25.10 Tile

Listing 109: Tile.java

```
package Model.Map;

import Model.GameObject.GameObject;
import Model.GameObject.ManaCrystal;
import Model.GameObject.Obstacle;
import static Resource.Constants.GeneralConstants.MapConstants.*;

/**
 * The Tile stores information and provides no functionality. They are organized
 * in a map and managed by the MapController.
 */
public class Tile implements Comparable<Tile>
{

    /**
     * Transform (Coordinates) of this Tile
     */
    private AxialTransform _axialTransform = null;

    /**
     * GameObject on the Tile
     */
    private GameObject _gameObject = null;

    /**
     * Tile's _height
     */
    private int _height = MIN_TILE_HEIGHT;

    /**
     * Priority is used for Pathfinding
     */
    private int _priority = DEFAULT_PRIORITY;

    /**
     * For selecting a tile and marking it in GUI
     */
    private boolean _isSelected = false;

    /**
     * for highlighting a tile for moves and marking it in GUI
     */
    private boolean _isHighlightedForMove = false;

    /**
     * for highlighting a tile for attacks and marking it in GUI
     */
    private boolean _isHighlightedForAttack = false;

    /**
     * the amount of movement points moving on this tile costs. since it is 1
     * for all tiles it is basically a constant but could be used if desired
     */
    private int _movementPointCost = DEFAULT_MOVEMENT_POINT_COST;

    /**
     * Constructs a new Tile with given axial coordinates and _height.
     *
     * @param t      Transform of the Tile
     * @param height Height of the Tile
     */
    protected Tile (AxialTransform t, int height)
```

```
    {
        this._axialTransform = t;
        this._height = height;
    }

    /**
     * Saves a Tile as a string for storing it in a .map file by the
     * MapGenerator Class.
     *
     * @return String representation of this tile
     *         [h,d,_height,objectHere,CrystalHere]
     */
    protected String saveToString ()
    {
        StringBuilder sb = new StringBuilder();
        sb.append(this._axialTransform.toString());
        sb.append(',');
        sb.append(this._height);
        sb.append(',');
        if (Obstacle.class.isInstance(this._gameObject))
          {
            sb.append('1');
          }
        else
          {
            if (ManaCrystal.class.isInstance(this._gameObject))
              {
                sb.append('2');
              }
            else
              {
                sb.append('0');
              }
          }

        sb.append("\n");
        return sb.toString();
    }

    /**
     *
     * @return this Tile's axial Transform.
     */
    public AxialTransform getAxialTransform ()
    {
        return this._axialTransform;
    }

    /**
     *
     * @return GameObject on this Tile.
     */
    public GameObject getGameObject ()
    {
        return this._gameObject;
    }

    /**
     *
     * @param o GameObject to place on this tile.
     */
    public void setGameObject (GameObject o)
    {
        this._gameObject = o;
    }
```

```
    /**
     * Setter
     *
     * @param priority new _priority.
     */
    public void setPriority (int priority)
    {
        this._priority = priority;
    }

    /**
     *
     * @return this Tile's _height.
     */
    public int getHeight ()
    {
        return this._height;
    }

    /**
     *
     * @return this tile's current _priority.
     */
    public int getPriority ()
    {
        return this._priority;
    }

    /**
     *
     * @return this Tile's movement point cost.
     */
    public int getMovementPointCost ()
    {
        return this._movementPointCost;
    }

    /**
     *
     * @return true, if this tile is currently highlighted for moves.
     */
    public boolean isSelected ()
    {
        return this._isSelected;
    }

    /**
     *
     * @param isSelected if this tile is selected or not.
     */
    public void setIsSelected (boolean isSelected)
    {
        this._isSelected = isSelected;
    }

    /**
     *
     * @return true, if this tile is currently highlighted for moves.
     */
    public boolean isHighlightedForMove ()
    {
        return this._isHighlightedForMove;
    }

    /**
     *
```

```
     * @param isHighlighted if this tile is highlighted for attack or not.
     */
    public void setIsHighlightedForMove (boolean isHighlighted)
    {
        this._isHighlightedForMove = isHighlighted;
    }

    /**
     *
     * @return true, if this tile is currently highlighted for attacks.
     */
    public boolean isHighlightedForAttack ()
    {
        return this._isHighlightedForAttack;
    }

    /**
     *
     * @param isHighlighted if this tile is highlighted for attack or not.
     */
    public void setIsHighlightedForAttack (boolean isHighlighted)
    {
        this._isHighlightedForAttack = isHighlighted;
    }

    /**
     * compares the tile with another tile by _priority.
     *
     * @param other Tile to compare this tile's _priority to.
     *
     * @return true if both tiles have the same _priority.
     */
    public boolean equals (Tile other)
    {
        return (this._priority == other.getPriority());
    }

    /**
     *
     * @param other The Tile this tile is to be compared to.
     *
     * @return 0, if the tiles _priority is equal, 1, if this tiles _priority is
     *         lower than the other's, -1, if this tiles _priority is higher
     *         than
     *         the other
     */
    @Override
    public int compareTo (Tile other)
    {
        if (this.equals(other))
          {
            return 0;
          }
        else if (this._priority > other.getPriority())
          {
            return 1;
          }
        else
          {
            return -1;
          }
    }

    /**
     *
     * @return HashCode of this Tile's Axial Transform.
```

```
     */
    @Override
    public int hashCode ()
    {
        return this._axialTransform.hashCode();
    }

    /**
     *
     * @return String of this Tile's Axial Transform.
     */
    @Override
    public String toString ()
    {
        return this._axialTransform.toString();
    }

}
```

6.26 Resource

6.26.1 Dimensions

Listing 110: Dimensions.java

```
package Resource;

import java.awt.Dimension;
import java.awt.Toolkit;

/**
 * this class exists to provide Information about the Users Screen Resolution.
 */
public class Dimensions
{

    /**
     * the User screens size stored as a Dimension.
     */
    private static Dimension _screenSize = Toolkit.getDefaultToolkit().getScreenSize();

    /**
     * Provides the Users Horizontal Screen Resolution.
     *
     * @return the horizontal screen resolution in Pixels.
     */
    public static int getWidth ()
    {
        return (int) _screenSize.getWidth();
    }

    /**
     * Provides the Users Vertical Screen Resolution.
     *
     * @return the vertical screen resolution in Pixels.
     */
    public static int getHeight ()
    {
        return (int) _screenSize.getHeight();
    }

}
```

6.26.2 VideoOptions

Listing 111: VideoOptions.java

```
package Resource;

/**
 * The default render resolution height and width are stored in this interface.
 */
public interface VideoOptions
{

    /**
     * Vertical Resolution at which the Game is being rendered and then scaled
     * down from
     *
     * Default Resolution: 4K 16:9.
     */
    public final static int DEFAULT_RENDER_RESOLUTION_HEIGHT = 2160;

    /**
     * Horizontal Resolution at which the Game is being rendered and then scaled
     * down from
     *
     * Default Resolution: 4K 16:9.
     */
    public final static int DEFAULT_RENDER_RESOLUTION_WIDTH = 3840;

}
```

6.27 Resource.Constants.CardConstants

6.27.1 CardConstants

Listing 112: CardConstants.java

```
package Resource.Constants.CardConstants;

/**
 * This interface stores all Constants that are generally related to cards.
 */
public interface CardConstants
{
    /*--------------------------Start-Ints--------------------------------*/

    /**
     *
     */
    public static final int DEFAULT_MAX_TARGETS = 1;

    /*--------------------------End-Ints----------------------------------*/
    /*--------------------------Start-Strings-----------------------------*/

    /**
     *
     */
    public static final String DEFAULT_CARD_DESCRIPTION = "No Description Available.";

    /**
     *
     */
    public static final String DEFAULT_CARD_EFFECT_NAME = "Unnamed Card Effect";

    /**
     *
     */
    public static final String DEFAULT_WAND_NAME = "Unnamed Wand";
```

```
    /*-------------------------End-Strings-----------------------------------*/
}
```

6.28 Resource.Constants.CardConstants.SpellConstants.Chaos

6.28.1 VengeanceSpellConstants

Listing 113: VengeanceSpellConstants.java

```
package Resource.Constants.CardConstants.SpellConstants.Chaos;

/**
 * These are all Constants relevant for initializing the spell: vengeance
 */
public interface VengeanceSpellConstants
{
    /*
     * -------------------Start-Ints-------------------------------------------
     */
    /**
     *
     */
    public static final int VENGEANCE_SPELL_CARD_ID = 20;

    /**
     *
     */
    public static final int VENGEANCE_SPELL_CAST_RANGE = 10;

    /**
     *
     */
    public static final int VENGEANCE_SPELL_MANA_COST = 20;

    /**
     *
     */
    public static final int VENGEANCE_SPELL_FLUX_CHANGE = 2;

    /**
     *
     */
    public static final int VENGEANCE_SPELL_CARD_EFFECT_ATTACK_VALUE = 80;

    /*
     * -------------------End---Ints-------------------------------------------
     */
 /*
     * -------------------Start-Strings----------------------------------------
     */
    /**
     *
     */
    public static final String VENGEANCE_SPELL_CARD_NAME = "Vengeance";

    /**
     *
     */
    public static final String VENGEANCE_SPELL_CARD_DESCRIPTION = "Deals 80 damage " +
            "to a enemy Creature. If the target is a enemy Wizard, doesn't deal" +
            " any damage and instead discards 2 cards from the enemies hand.";
```

```
    /**
     *
     */
    public static final String VENGEANCE_SPELL_CARD_EFFECT_SPELL_NAME =
            "Vengeance Effect";

    /*
     * -------------------End-Strings-----------------------------------------
     */
}
```

6.29 Resource.Constants.CardConstants.SpellConstants.Law

6.29.1 DecreeSpellConstants

Listing 114: DecreeSpellConstants.java

```
package Resource.Constants.CardConstants.SpellConstants.Law;

/**
 * These are all Constants relevant for initializing the spell: Decree
 */
public interface DecreeSpellConstants
{
    /*
     * ----------------------------------Start-Ints--------------------------
     */

    /**
     *
     */
    public static final int DECREE_SPELL_CARD_ID = 16;

    /**
     *
     */
    public static final int DECREE_SPELL_CAST_RANGE = 10;

    /**
     *
     */
    public static final int DECREE_SPELL_MANA_COST = 20;

    /**
     *
     */
    public static final int DECREE_SPELL_FLUX_CHANGE = 2;

    /**
     *
     */
    public static final int DECREE_SPELL_CARD_EFFECT_ATTACK_VALUE = 80;

    /**
     *
     */
    public static final int DECREE_SPELL_ON_CAST_AMOUNT_OF_DISCARD = 1;

    /**
     *
     */
    public static final int DECREE_SPELL_ON_CAST_AMOUNT_OF_DISCARD_IS_NOT_MULTIPLAYER
            = 1;

    /*
```

```
     * ----------------------------------End---Ints--------------------------
     */
 /*
     * ----------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public static final String DECREE_SPELL_CARD_NAME = "Decree";

    /**
     *
     */
    public static final String DECREE_SPELL_CARD_EFFECT_SPELL_NAME =
            "Justice Effect";

    /**
     *
     */
    public static final String DECREE_SPELL_CARD_DESCRIPTION = "Deals 80 damage "
            + "to a " +
            "enemy Creature. If the target is a enemy Wizard, doesn't deal " +
            "any damage and instead discards 2 cards from the enemies hand.";

    /*
     * ----------------------------------End-Strings-------------------------
     */
}
```

6.29.2 DivineBowSpellConstants

Listing 115: DivineBowSpellConstants.java

```
package Resource.Constants.CardConstants.SpellConstants.Law;

/**
 * These are all Constants relevant for initializing the spell: DivineBow
 */
public interface DivineBowSpellConstants
{
    /*
     * ----------------------------------Start-Ints--------------------------
     */

    /**
     *
     */
    public static final int DIVINE_BOW_SPELL_CARD_ID = 17;

    /**
     *
     */
    public static final int DIVINE_BOW_SPELL_MANA_COST = 40;

    /**
     *
     */
    public static final int DIVINE_BOW_SPELL_FLUX_CHANGE = 3;

    /**
     *
     */
    public static final int DIVINE_BOW_SPELL_CAST_RANGE = 0;

    /**
     *
```

```
     */
    public static final int DIVINE_BOW_SPELL_ON_CAST_SET_ATTACK_RANGE = 3;

    /**
     *
     */
    public static final int DIVINE_BOW_SPELL_ON_CAST_ADD_RANGED_ATTACK = 20;

    /*
     * -----------------------------------End---Ints--------------------------
     */
  /*
     * -----------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public static final String DIVINE_BOW_SPELL_CARD_NAME = "Divine Bow";

    /**
     *
     */
    public static final String DIVINE_BOW_SPELL_CARD_DESCRIPTION = "When cast" +
            " on the friendly Wizard, grants + 20 to Ranged Attack and + 3 to" +
            " Attack Range.";

    /*
     * -----------------------------------End---Strings-----------------------
     */
}
```

6.29.3 DivineShieldSpellConstants

Listing 116: DivineShieldSpellConstants.java

```
package Resource.Constants.CardConstants.SpellConstants.Law;

/**
 * These are all Constants relevant for initializing the spell: Divine Shield
 */
public interface DivineShieldSpellConstants
{
    /*
     * -----------------------Start-Ints-------------------------------------
     */

    /**
     *
     */
    public static final int DIVINE_SHIELD_SPELL_CARD_ID = 18;

    /**
     *
     */
    public static final int DIVINE_SHIELD_SPELL_MANA_COST = 35;

    /**
     *
     */
    public static final int DIVINE_SHIELD_SPELL_FLUX_CHANGE = 3;

    /**
     *
     */
    public static final int DIVINE_SHIELD_SPELL_CAST_RANGE = 0;
```

```
    /**
     *
     */
    public static final int DIVINE_SHIELD_SPELL_ON_CAST_ADD_PHYS_RES = 20;

    /*
     * -----------------------End---Ints---------------------------------------
     */
 /*
     * -----------------------Start-Strings-----------------------------------
     */
    /**
     *
     */
    public static final String DIVINE_SHIELD_SPELL_CARD_NAME = "Divine Shield";

    /**
     *
     */
    public static final String DIVINE_SHIELD_SPELL_CARD_DESCRIPTION =
        "When cast on the friendly Wizard, grants + 20 to physical Resistance";

    /*
     * -----------------------End-Strings-------------------------------------
     */
}
```

6.29.4 DivineSwordSpellConstants

Listing 117: DivineSwordSpellConstants.java

```
package Resource.Constants.CardConstants.SpellConstants.Law;

/**
 * These are all Constants relevant for initializing the spell: DivineSword
 */
public interface DivineSwordSpellConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int DIVINE_SWORD_SPELL_CARD_ID = 19;

    /**
     *
     */
    public static final int DIVINE_SWORD_SPELL_MANA_COST = 30;

    /**
     *
     */
    public static final int DIVINE_SWORD_SPELL_FLUX_CHANGE = 3;

    /**
     *
     */
    public static final int DIVINE_SWORD_SPELL_CAST_RANGE = 0;

    /**
     *
     */
    public static final int DIVINE_SWORD_SPELL_ON_CAST_ADD_ATTACK = 90;
```

```
    /*
     * -----------------------------------End---Ints-------------------------
     */
 /*
     * -----------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public static final String DIVINE_SWORD_SPELL_CARD_NAME = "Divine Sword";

    /**
     *
     */
    public static final String DIVINE_SWORD_SPELL_CARD_DESCRIPTION =
        "When cast on the friendly Wizard, grants + 80 to Attack.";

    /*
     * -----------------------------------End---Strings----------------------
     */
}
```

6.30 Resource.Constants.CardConstants.SpellConstants.Neutral

6.30.1 MagicBoltSpellConstants

Listing 118: MagicBoltSpellConstants.java

```
package Resource.Constants.CardConstants.SpellConstants.Neutral;

import static Resource.Constants.MinionConstants.PlayerCharacterMinionConstants.
        DEFAULT_PLAYER_CHARACTER_MAGIC_RES_VALUE;

/**
 * These are all Constants relevant for initializing the spell: MagicBolt
 */
public interface MagicBoltSpellConstants
{
    /*
     * -----------------------------------Start-booleans---------------------
     */

    /**
     *
     */
    public static final boolean MAGIC_BOLT_SPELL_REQUIRES_LINE_OF_SIGHT = true;

    /*
     * -----------------------------------End-booleans-----------------------
     */
 /*
     * -----------------------------------Start-Ints-------------------------
     */
    /**
     *
     */
    public static final int MAGIC_BOLT_SPELL_CARD_ID = 14;

    /**
     *
     */
    public static final int MAGIC_BOLT_SPELL_CAST_RANGE = 3;

    /**
```

```
     *
     */
    public static final int MAGIC_BOLT_SPELL_CARD_EFFECT_SET_ATTACK_VALUE =
            DEFAULT_PLAYER_CHARACTER_MAGIC_RES_VALUE / 2 + 50;

    /*
     * ------------------------------------End---Ints-------------------------
     */
 /*
     * ------------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public static final String MAGIC_BOLT_SPELL_CARD_EFFECT_SET_SPELL_NAME =
            "Magic Bolt Effect";

    /**
     *
     */
    public static final String MAGIC_BOLT_SPELL_CARD_NAME = "Magic Bolt";

    /**
     *
     */
    public static final String MAGIC_BOLT_SPELL_CARD_DESCRIPTION = "When cast on" +
             " an enemy Creature, deals " + (DEFAULT_PLAYER_CHARACTER_MAGIC_RES_VALUE /
             2 + 50) + " Damage.";

    /*
     * ------------------------------------End---Stirngs----------------------
     */
}
```

6.30.2 WindWalkerSpellConstants

Listing 119: WindWalkerSpellConstants.java

```
package Resource.Constants.CardConstants.SpellConstants.Neutral;

/**
 * These are all Constants relevant for initializing the spell: WindWalker
 */
public interface WindWalkerSpellConstants
{
    /*
     * ------------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int WIND_WALKER_SPELL_CARD_ID = 15;

    /**
     *
     */
    public static final int WIND_WALKER_SPELL_MANA_COST = 40;

    /**
     *
     */
    public static final int WIND_WALKER_SPELL_FLUX_CHANGE = 0;

    /**
     *
```

```
     */
    public static final int WIND_WALKER_SPELL_CAST_RANGE = 0;

    /**
     *
     */
    public static final int WIND_WALKER_SPELL_ON_CAST_ADD_MAX_MVP = 1;

    /*
     * -----------------------------------End---Ints-------------------------
     */
 /*
     * -----------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public static final String WIND_WALKER_SPELL_CARD_NAME = "Wind Walker";

    /**
     *
     */
    public static final String WIND_WALKER_SPELL_CARD_DESCRIPTION = "When cast " +
            "on a friendly Wizard, grants + 1 to movement range as well as " +
            "the ability to fly!";

    /*
     * -----------------------------------End-Strings------------------------
     */
}
```

6.31 Resource.Constants.CardConstants.SummonConstants

6.31.1 SummonConstants

Listing 120: SummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants;

/**
 * These are all constants relevant for all summons.
 */
public interface SummonConstants
{
    /*
     * -----------------------Start-Ints---------------------------------------
     */

    /**
     *
     */
    public final static int DEFAULT_FLUX_CHANGE = 0;

    /*
     * -----------------------End-Ints-----------------------------------------
     */
 /*
     * -----------------------Start-Strings------------------------------------
     */
    /**
     *
     */
    public static final String SUMMON_EFFECT_CARD_NAME = " Effect";

    /*
```

```
     * -----------------------End-Strings-------------------------------------
     */
}
```

6.32 Resource.Constants.CardConstants.SummonConstants.Chaos

6.32.1 HellhoundSummonConstants

Listing 121: HellhoundSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Chaos;

/**
 * These are all Constants relevant for initializing the summon: Hellhound
 */
public interface HellhoundSummonConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int HELLHOUND_SUMMON_CARD_ID = 11;

    /**
     *
     */
    public final static int HELLHOUND_SUMMON_MANA_COST = 65;

    /**
     *
     */
    public final static int HELLHOUND_SUMMON_FLUX_CHANGE = 4;

    /*
     * -----------------------------------End---Ints-------------------------
     */
 /*
     * -----------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String HELLHOUND_SUMMON_CARD_NAME = "Summon Hellhound";

    /**
     *
     */
    public final static String HELLHOUND_SUMMON_CARD_DESCRIPTION =
            "Summons a Hellhound on the target location.";

    /*
     * -----------------------------------End--Strings-----------------------
     */
}
```

6.32.2 HydraSummonConstants

Listing 122: HydraSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Chaos;
```

```
/**
 * These are all Constants relevant for initializing the summon: Hydra
 */
public interface HydraSummonConstants
{
    /*
     * ------------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int HYDRA_SUMMON_CARD_ID = 12;

    /**
     *
     */
    public final static int HYDRA_SUMMON_MANA_COST = 70;

    /**
     *
     */
    public final static int HYDRA_SUMMON_FLUX_CHANGE = 5;

    /*
     * ------------------------------------End---Ints-------------------------
     */
 /*
     * ------------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String HYDRA_SUMMON_CARD_NAME = "Summon Hydra";

    /**
     *
     */
    public final static String HYDRA_SUMMON_CARD_DESCRIPTION =
            "Summons a Hydra on the target location.";

    /*
     * ------------------------------------End---Strings----------------------
     */
}
```

6.32.3 ManticoreSummonConstants

Listing 123: ManticoreSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Chaos;

/**
 * These are all Constants relevant for initializing the summon: Manticore
 */
public interface ManticoreSummonConstants
{
    /*
     * ------------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int MANTICORE_SUMMON_CARD_ID = 8;
```

```
    /**
     *
     */
    public final static int MANTICORE_SUMMON_MANA_COST = 60;

    /**
     *
     */
    public final static int MANTICORE_SUMMON_FLUX_CHANGE = 4;

    /*
     * ------------------------------------End---Ints-------------------------
     */
 /*
     * ------------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String MANTICORE_SUMMON_CARD_NAME = "Summon Manticore";

    /**
     *
     */
    public final static String MANTICORE_SUMMON_CARD_DESCRIPTION =
            "Summons a Manticore on the target location.";

    /*
     * ------------------------------------End---Strings----------------------
     */
}
```

6.32.4 SkeletonSummonConstants

Listing 124: SkeletonSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Chaos;

/**
 * These are all Constants relevant for initializing the summon: Skeleton
 */
public interface SkeletonSummonConstants
{
    /*
     * ------------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int SKELETON_SUMMON_CARD_ID = 10;

    /**
     *
     */
    public final static int SKELETON_SUMMON_MANA_COST = 25;

    /**
     *
     */
    public final static int SKELETON_SUMMON_FLUX_CHANGE = 3;

    /*
     * ------------------------------------End---Ints-------------------------
     */
 /*
```

```
     * ------------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public final static String SKELETON_SUMMON_CARD_NAME = "Summon Skeleton";

    /**
     *
     */
    public final static String SKELETON_SUMMON_CARD_DESCRIPTION =
            "Summons a Skeleton on the target location.";

    /*
     * ------------------------------------End---Strings-----------------------
     */
}
```

6.32.5 VampireSummonConstants

Listing 125: VampireSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Chaos;

/**
 * These are all Constants relevant for initializing the summon: Vampire
 */
public interface VampireSummonConstants
{
    /*
     * ------------------------------------Start-Ints--------------------------
     */

    /**
     *
     */
    public static final int VAMPIRE_SUMMON_CARD_ID = 9;

    /**
     *
     */
    public final static int VAMPIRE_SUMMON_MANA_COST = 60;

    /**
     *
     */
    public final static int VAMPIRE_SUMMON_FLUX_CHANGE = 4;

    /*
     * ------------------------------------End---Ints--------------------------
     */
 /*
     * ------------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public final static String VAMPIRE_SUMMON_CARD_NAME = "Summon Vampire";

    /**
     *
     */
    public final static String VAMPIRE_SUMMON_CARD_DESCRIPTION =
            "Summons a Vampire on the target location.";

    /*
```

```
     * ------------------------------------End---Strings-----------------------
     */
}
```

6.32.6 ZombieSummonConstants

Listing 126: ZombieSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Chaos;

/**
 * These are all Constants relevant for initializing the summon: Zombie
 */
public interface ZombieSummonConstants
{
    /*
     * ------------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int ZOMBIE_SUMMON_CARD_ID = 13;

    /**
     *
     */
    public final static int ZOMBIE_SUMMON_MANA_COST = 35;

    /**
     *
     */
    public final static int ZOMBIE_SUMMON_FLUX_CHANGE = 2;

    /*
     * ------------------------------------End---Ints-------------------------
     */
 /*
     * ------------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String ZOMBIE_SUMMON_CARD_NAME = "Summon Zombie";

    /**
     *
     */
    public final static String ZOMBIE_SUMMON_CARD_DESCRIPTION =
            "Summons a Zombie on the target location.";

    /*
     * ------------------------------------End---Strings----------------------
     */
}
```

6.33 Resource.Constants.CardConstants.Law

6.33.1 IcarusTowerSummonConstants

Listing 127: IcarusTowerSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Law;
```

```
/**
 * These are all Constants relevant for initializing the summon: IcarusTower
 */
public interface IcarusTowerSummonConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int ICARUS_TOWER_SUMMON_CARD_ID = 5;

    /**
     *
     */
    public final static int ICARUS_TOWER_SUMMON_MANA_COST = 40;

    /**
     *
     */
    public final static int ICARUS_TOWER_SUMMON_FLUX_CHANGE = 3;

    /*
     * -----------------------------------End---Ints-------------------------
     */
 /*
     * -----------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String ICARUS_TOWER_SUMMON_CARD_NAME =
            "Summon Icarus Tower";

    /**
     *
     */
    public final static String ICARUS_TOWER_SUMMON_CARD_DESCRIPTION =
            "Summons an Icarus Tower on the target location.";

    /*
     * -----------------------------------End---Strings----------------------
     */
}
```

6.33.2 PaladinSummonConstants

Listing 128: PaladinSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Law;

/**
 * These are all Constants relevant for initializing the summon: Paladin
 */
public interface PaladinSummonConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int PALADIN_SUMMON_CARD_ID = 7;
```

```
    /**
     *
     */
    public final static int PALADIN_SUMMON_MANA_COST = 60;

    /**
     *
     */
    public final static int PALADIN_SUMMON_FLUX_CHANGE = 4;

    /*
     * -----------------------------------End---Ints-------------------------
     */
 /*
     * -----------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String PALADIN_SUMMON_CARD_NAME = "Summon Paladin";

    /**
     *
     */
    public final static String PALADIN_SUMMON_CARD_DESCRIPTION =
            "Summons a Paladin on the target location.";

    /*
     * -----------------------------------End---Strings----------------------
     */
}
```

6.33.3 PegasusSummonConstants

Listing 129: PegasusSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Law;

/**
 * These are all Constants relevant for initializing the summon: Pegasus
 */
public interface PegasusSummonConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int PEGASUS_SUMMON_CARD_ID = 6;

    /**
     *
     */
    public final static int PALADIN_SUMMON_MANA_COST = 60;

    /**
     *
     */
    public final static int PALADIN_SUMMON_FLUX_CHANGE = 4;

    /*
     * -----------------------------------End---Ints-------------------------
     */
```

```
 /*
     * -----------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public final static String PALADIN_SUMMON_CARD_NAME = "Summon Paladin";

    /**
     *
     */
    public final static String PALADIN_SUMMON_CARD_DESCRIPTION =
            "Summons a Paladin on the target location.";

    /*
     * -----------------------------------End---Strings-----------------------
     */
}
```

6.33.4 SapphireDragonSummonConstants

Listing 130: SapphireDragonSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Law;

/**
 * These are all Constants relevant for initializing the summon: SapphireDragon
 */
public interface SapphireDragonSummonConstants
{
    /*
     * -----------------------------------Start-Ints--------------------------
     */

    /**
     *
     */
    public static final int SAPPHIRE_DRAGON_SUMMON_CARD_ID = 4;

    /**
     *
     */
    public final static int SAPPHIRE_DRAGON_SUMMON_MANA_COST = 80;

    /**
     *
     */
    public final static int SAPPHIRE_DRAGON_SUMMON_FLUX_CHANGE = 5;

    /*
     * -----------------------------------End---Ints--------------------------
     */
 /*
     * -----------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public final static String SAPPHIRE_DRAGON_SUMMON_CARD_NAME =
            "Summon Sapphire Dragon";

    /**
     *
     */
    public final static String SAPPHIRE_DRAGON_SUMMON_CARD_DESCRIPTION =
            "Summons an Sapphire Dragon on the target location.";
```

```
    /*
     * -----------------------------------End---Strings-----------------------
     */
}
```

6.34 Resource.Constants.CardConstants.Neutral

6.34.1 AirElementalSummonConstants

Listing 131: AirElementalSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Neutral;

/**
 * These are all Constants relevant for initializing the summon: AirElemental
 */
public interface AirElementalSummonConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int AIR_ELEMENTAL_SUMMON_CARD_ID = 2;

    /**
     *
     */
    public final static int AIR_ELEMENTAL_SUMMON_MANA_COST = 30;

    /*
     * -----------------------------------End---Ints-------------------------
     */
  /*
     * -----------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String AIR_ELEMENTAL_SUMMON_CARD_NAME =
            "Summon Air Elemental";

    /**
     *
     */
    public final static String AIR_ELEMENTAL_SUMMON_CARD_DESCRIPTION =
            "Summons an Air Elemental on the target location.";

    /*
     * -----------------------------------End---Strings----------------------
     */
}
```

6.34.2 EagleSummonConstants

Listing 132: EagleSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Neutral;

/**
 * These are all Constants relevant for initializing the summon: Eagle
```

```
 */
public interface EagleSummonConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int EAGLE_SUMMON_CARD_ID = 1;

    /**
     *
     */
    public final static int EAGLE_SUMMON_MANA_COST = 20;

    /*
     * -----------------------------------End---Ints-------------------------
     */
 /*
     * -----------------------------------Start-Strings----------------------
     */
    /**
     *
     */
    public final static String EAGLE_SUMMON_CARD_NAME = "Summon Eagle";

    /**
     *
     */
    public final static String EAGLE_SUMMON_CARD_DESCRIPTION =
            "Summons an Eagle on the target location.";

    /*
     * -----------------------------------End---Strings----------------------
     */
}
```

6.34.3 GiantSummonConstants

Listing 133: GiantSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Neutral;

/**
 * These are all Constants relevant for initializing the summon: Giant
 */
public interface GiantSummonConstants
{
    /*
     * -----------------------------------Start-Ints-------------------------
     */

    /**
     *
     */
    public static final int GIANT_SUMMON_CARD_ID = 3;

    /**
     *
     */
    public final static int GIANT_SUMMON_MANA_COST = 60;

    /*
     * -----------------------------------End---Ints-------------------------
```

```
     */
 /*
     * -----------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public final static String GIANT_SUMMON_CARD_NAME = "Summon Giant";

    /**
     *
     */
    public final static String GIANT_SUMMON_CARD_DESCRIPTION =
            "Summons an Giant on the target location.";

    /*
     * -----------------------------------End---Strings-----------------------
     */
}
```

6.34.4 LionSummonConstants

Listing 134: LionSummonConstants.java

```
package Resource.Constants.CardConstants.SummonConstants.Neutral;

/**
 * These are all Constants relevant for initializing the summon: Lion
 */
public interface LionSummonConstants
{
    /*
     * -----------------------------------Start-Ints--------------------------
     */

    /**
     *
     */
    public static final int LION_SUMMON_CARD_ID = 0;

    /**
     *
     */
    public final static int LION_SUMMON_MANA_COST = 30;

    /*
     * -----------------------------------End---Ints--------------------------
     */
 /*
     * -----------------------------------Start-Strings-----------------------
     */
    /**
     *
     */
    public final static String LION_SUMMON_CARD_NAME = "Summon Lion";

    /**
     *
     */
    public final static String LION_SUMMON_CARD_DESCRIPTION =
            "Summons an Lion on the target location.";

    /*
     * -----------------------------------End---Strings-----------------------
     */
}
```

6.35 Resource.Constants.CardConstants.WandConstants

6.35.1 AirWandConstants

Listing 135: AirWandConstants.java

```java
package Resource.Constants.CardConstants.WandConstants;

/**
 * The AirWandConstants contain all constant values related to the AirWand,
 * for example the amount of the specific cards that the airwand contains.
 */
public interface AirWandConstants
{
    /*----------------------------Start-Ints----------------------------------*/

    /**
     * The amount SapphireDragon cards in the AirWand.
     */
    public static final int AMOUNT_OF_SAPPHIRE_DRAGON_AIR_WAND = 5;

    /**
     * The amount Pegasus cards in the AirWand.
     */
    public static final int AMOUNT_OF_PEGASUS_AIR_WAND = 3;

    /**
     * The amount Vampire cards in the AirWand.
     */
    public static final int AMOUNT_OF_VAMPIRE_AIR_WAND = 3;

    /**
     * The amount Manticore cards in the AirWand.
     */
    public static final int AMOUNT_OF_MANTICORE_AIR_WAND = 3;

    /**
     * The amount Eagle cards in the AirWand.
     */
    public static final int AMOUNT_OF_EAGLE_AIR_WAND = 3;

    /**
     * The amount IcarusTower cards in the AirWand.
     */
    public static final int AMOUNT_OF_ICARUS_TOWER_AIR_WAND = 3;

    /**
     * The amount WindWalker cards in the AirWand.
     */
    public static final int AMOUNT_OF_WIND_WALKER_AIR_WAND = 3;

    /**
     * The amount AirElemental cards in the AirWand.
     */
    public static final int AMOUNT_OF_AIR_ELEMENTAL_AIR_WAND = 3;

    /**
     * The amount MagicBolt cards in the AirWand.
     */
    public static final int AMOUNT_OF_MAGIC_BOLT_AIR_WAND = 4;


    /*----------------------------End-Ints------------------------------------*/
    /*----------------------------Start-Strings-------------------------------*/
```

```
    /**
     * The name of the AirWand for tooltips etc.
     */
    public static final String AIR_WAND_NAME = "Air Wand";

    /*----------------------------End-Strings--------------------------------*/
}
```

6.35.2 CreatureWandConstants

Listing 136: CreatureWandConstants.java

```
package Resource.Constants.CardConstants.WandConstants;

/**
 * The CreatureWandConstants contain all the Constants about the CreatureWand,
 * for example the amount of the specific cards in the CreatureWand.
 */
public interface CreatureWandConstants
{
    /*--------------------Start---ints----------------------------------------*/

    /**
     * The amount of Vengeance Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_VENGEANCE_CREATURE_WAND = 1;

    /**
     * The amount of Decree Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_DECREE_CREATURE_WAND = 1;

    /**
     * The amount of DivineBow Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_DIVINE_BOW_CREATURE_WAND = 3 ;

    /**
     * The amount of SapphireDragon Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_SAPPHIRE_DRAGON_CREATURE_WAND = 4;

    /**
     * The amount of Paladin Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_PALADIN_CREATURE_WAND = 2;

    /**
     * The amount of Giant Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_GIANT_CREATURE_WAND = 2;

    /**
     * The amount of Lion Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_LION_CREATURE_WAND = 3;

    /**
     * The amount of Hydra Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_HYDRA_CREATURE_WAND = 3;

    /**
     * The amount of vampire Cards in the CreatureWand.
     */
```

```
    public static final int AMOUNT_OF_VAMPIRE_CREATURE_WAND = 3;

    /**
     * The amount of DivineShield Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_DIVINE_SHIELD_CREATURE_WAND = 2;

    /**
     * The amount of DivineSword Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_DIVINE_SWORD_CREATURE_WAND = 2;

    /**
     * The amount of MagicBolt Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_MAGIC_BOLT_CREATURE_WAND = 2;

    /**
     * The amount of AirElemental Cards in the CreatureWand.
     */
    public static final int AMOUNT_OF_AIR_ELEMENTAL_CREATURE_WAND = 2;

    /*--------------------End---ints-----------------------------------------*/
    /*--------------------Start-Strings--------------------------------------*/

    /**
     * The name of the CreatureWand for Tooltips etc.
     */
    public static final String CREATURE_WAND_NAME = "Creature Wand";

    /*--------------------End---Strings--------------------------------------*/
}
```

6.35.3 RandomWandConstants

Listing 137: RandomWandConstants.java

```
package Resource.Constants.CardConstants.WandConstants;

/**
 * These are all Constants relevant for initializing a Random Wand
 */
public class RandomWandConstants
{
    /*
     * ---------------------------Start-Strings-------------------------------
     */

    /**
     *
     */
    public static final String RANDOM_WAND_NAME = "Random Wand";

    /**
     *
     */
    public static final String RANDOM_WAND_TOOLTIP_INFO_WAND = "Wand";

    /**
     *
     */
    public static final String RANDOM_WAND_TOOLTIP_INFO_RANDOM = "Random";

    /*
     * ----------------------------End-Strings-------------------------------
     */
```

```
}
```

6.35.4 UndeadWandConstants

Listing 138: UndeadWandConstants.java

```
package Resource.Constants.CardConstants.WandConstants;

/**
 * The UndeadWandConstants contain all constant values that are related to the
 * UndeadWand, for example the amount of the specific cards that the UndeadWand
 * contains.
 */
public interface UndeadWandConstants
{

    /*--------------Start-Ints-----------------------------------------------*/

    /**
     * The amount of Lion cards in the UndeadWand.
     */
    public static final int AMOUNT_OF_LION_UNDEAD_WAND = 5;

    /**
     * The amount of Vampire cards in the UndeadWand.
     */
    public static final int AMOUNT_OF_VAMPIRE_UNDEAD_WAND = 6;

    /**
     * The amount of Skeleton cards in the UndeadWand.
     */
    public static final int AMOUNT_OF_SKELETON_UNDEAD_WAND = 6;

    /**
     * The amount of Hellhound cards in the UndeadWand.
     */
    public static final int AMOUNT_OF_HELLHOUND_UNDEAD_WAND = 6;

    /**
     * The amount of Decree cards in the UndeadWand.
     */
    public static final int AMOUNT_OF_DECREE_UNDEAD_WAND = 1;

    /**
     * The amount of Vengeance cards in the UndeadWand.
     */
    public static final int AMOUNT_OF_VENGEANCE_UNDEAD_WAND = 1;

    /**
     * The amount of Zombie cards in the UndeadWand.
     */
    public static final int AMOUNT_OF_ZOMBIE_UNDEAD_WAND = 5;

    /*--------------End-Ints-------------------------------------------------*/
    /*--------------Start-Strings--------------------------------------------*/

    /**
     * The name of the UndeadWand for tooltips etc.
     */
    public static final String UNDEAD_WAND_NAME = "Undead Wand";

    /*--------------End---Strings--------------------------------------------*/
}
```

6.36 Resource.Constants.GeneralConstants

6.36.1 ActionConstants

Listing 139: ActionConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 *
 */
public interface ActionConstants
{
    /*---------------------------Start-ints-----------------------------------*/

    /**
     *
     */
    public static final int SECOND_LAST_TILE = 2;

    /**
     *
     */
    public static final int ATTACKER_LIVES = -1;

    /*---------------------------Start-ints-----------------------------------*/
    /*---------------------------Start-floats---------------------------------*/

    /**
     *
     */
    public static final float DIVIDE_WITH_HUNDRED = 100f;

    /*---------------------------End---floats---------------------------------*/
}
```

6.36.2 ControllerConstants

Listing 140: ControllerConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 *
 */
public interface ControllerConstants
{
    /*------------------Start---MappingConstants-----------------------------*/

    /*----------------------Start-ints-----------------------------------------*/

    /**
     * must be smaller than radius and greater/equal 0
     */
    public static final int SPAWN_DISTANCE_FROM_EDGE = 1;

    /**
     *
     */
    public static final int NUMBER_OF_WIZARD_SPAWNS = 2;

    /**
     *
     */
```

```
    public static final int NUMBER_OF_PLAYERS = 2;

    /**
     *
     */
    public static final int MAXIMUM_OF_TRIES = 10;

    /**
     * The number of actions that can be reverted.
     */
    public static final int NUMBER_OF_ACTIONS_REVERTABLE = 10;

    /**
     * The default value for how much mana each player gains per turn.
     */
    public static final int DEFAULT_MANA_PER_TURN = 32;

    /*--------------------------End-ints--------------------------------------*/

    /*---------------------------Start-Strings--------------------------------*/

    /**
     *
     */
    public static final String TITLE = "ChaosRebornReborn";

    /**
     *
     */
    public static final String NO_CLIENT = "Client does not Exist.";

    /**
     *
     */
    public static final String NOT_MULTI = "Your game is not Multiplayer...";

    /**
     *
     */
    public static final String SPACE = " ";

    /**
     *
     */
    public static final String MOVE = "move ";

    /*---------------------------End-Strings----------------------------------*/
}
```

6.36.3 GameConstants

Listing 141: GameConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 *
 */
public interface GameConstants
{
    /*----------------------Start-ints----------------------------------------*/

    /**
     * The maximum flux shift towards chaos.
     */
    public static final int MAX_FLUX = 100;
```

```
    /**
     * The maximum flux shift towards law.
     */
    public static final int MIN_FLUX = -100;

     /**
     *
     */
    public static final int GAME_THREAD_SLEEP = 16;

    /*----------------------End---ints----------------------------------------*/
    /*----------------------Start-Strings-------------------------------------*/

    /**
     *
     */
    public static final String GAME_THREAD_NAME = "GameThread";

    /**
     *
     */
    public static final String GAME_THREAD_INTERRUPTED = " interrupted";

    /**
     *
     */
    public static final String TOOLTIP_DEFAULT_NAME = "Unidentified GameObject.";

    /*----------------------End-Strings---------------------------------------*/
}
```

6.36.4 GameObjectConstants

Listing 142: GameObjectConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 *
 */
public interface GameObjectConstants
{
    /*------------------------------Start-ints------------------------------*/

    /**
     *
     */
    public static final int DEFAULT_ATTACK_RANGE = 1;

    /**
     *
     */
    public static final int DEFAULT_ONETIMESHOT_ATTACK_RANGE = 1;

    /**
     *
     */
    public static final int DEFAULT_ONETIMESHOT_RANGED_ATTACK = 0;

    /**
     * The default size of the manaCrystals
     */
    public static final int DEFAULT_MANA_SIZE = 0;

    /**
```

```
     * The default value for mana crystals on the map
     */
    public static final int DEFAULT_MANA_CRYSTAL_GAIN = 10;

        /**
     * The Leech Attacker heal value
     */
    public static final int LEECH_ATTACKER_HEAL = 10;

    /**
     * The Leech bonus damage value
     */
    public static final int LEECH_BONUS_DAMAGE = 20;

    /**
     *
     */
    public static final int DEFAULT_OBSTACLE_SIZE = 2;

    /*-------------------------------End-ints---------------------------------*/
    /*-------------------------------Start-Strings----------------------------*/

    /**
     *
     */
    public static final String UNNAMED_TOOLTIP = "Unnamed Tooltip";

    /**
     *
     */
    public static final String MANA_CRYSTAL_TOOLTIP = "Mana Crystal";

    /**
     *
     */
    public static final String MANA_CRYSTAL_TOOLTIP_GRANTS = "Grants";

    /**
     *
     */
    public static final String MANA_CRYSTAL_TOOLTIP_MANA = " Mana";

    /**
     *
     */
    public static final String OBSTACLE_NAME = "Obstacle";


    /**
     *
     */
    public static final String SIZE = "Size";

    /*-------------------------------End-Strings------------------------------*/
}
```

6.36.5 MapConstants

Listing 143: MapConstants.java

```
package Resource.Constants.GeneralConstants;

import Model.Map.AxialTransform;
import Model.Map.MapGenerationKeywords;

/**
```

```
 *
 */
public interface MapConstants
{

    /*
     * -----------------------Start-Gen_Args----------------------------
     */
    /**
     * MapGenerationKeywords that are currently in use. First mus always be the
     * shape, second the size and third wether or not the map is randomized.
     */
    public static final MapGenerationKeywords[] DEFAULT_MAP_GEN_ARGS =
      {
        MapGenerationKeywords.hexagon, MapGenerationKeywords.medium,
        //   MapGenerationKeywords.random
        MapGenerationKeywords.noRandom
      };

    /**
     * When generating the map, each tile recieves a pseudorandom value between
     * 0 and 1. if the value is above DEFAULT_OBSTACLE_AMOUNT, the tile will
     * have an obstacle.
     */
    public static final float DEFAULT_OBSTACLE_AMOUNT = 0.80f;

    /**
     * When generating the map, each tile recieves a pseudorandom value between
     * 0 and 1. if the value is below DEFAULT_CRYSTAL_AMOUNT, the tile will have
     * a crystal.
     */
    public static final float DEFAULT_CRYSTAL_AMOUNT = 0.20f;

    /**
     * This makes the map height more or less rough by increasing or decreasing
     * the noise cell scale relative to the maps hexes. Recommended to be
     * between 0.3 and 0.8
     */
    public static final float TERRAIN_NOISE_SCALE = 0.4f;

    /**
     * See terrain_noise_scale but with obstacles and crystals. Recommended to
     * be between 0.8 and 1.0
     */
    public static final float FEATURE_NOISE_SCALE = 1.0f;

    /**
     * If the MapGenerationKeywords are invalid, use this map instead.
     */
    public static final String DEFAULT_MAP = "hexagon_medium";

    /**
     *
     */
    public static final String MAP_NAME_SMALL = "hexagon_small";

    /**
     *
     */
    public static final String MAP_NAME_MEDIUM = "hexagon_medium";

    /**
     *
     */
    public static final String MAP_NAME_LARGE = "hexagon_large";
```

```
    /**
     *
     */
    public static final String TILE_OVERLAP = "Tile Overlap";

    /**
     *
     */
    public static final String MAP_DATAFORMAT = ".map";

    /**
     *
     */
    public static final String FALLTHROUGH = "fallthrough";

    /*
     * ------------------------end-Gen_Args------------------------------------
     */
/*
     * ------------------------Start-ints----------------------------------
     */
    /**
     * Minimum height a tile generated by the mapGenerator will have.
     */
    public static final int MIN_TILE_HEIGHT = 1;

    /**
     * Maximum height a tile generated by the mapGenerator will have. According
     * to the task this will always be 3.
     */
    public static final int MAX_TILE_HEIGHT = 3;

    /**
     * Name outdated. Length of the edges of a Rhombus shaped map. Diameter of a
     * Hexagon shaped map. Named after use in Hexagonal maps because they will
     * likely be used more frequently ingame.
     */
    public static final int DIAMETER = 7;

    /**
     * How many Axial-Coordinates a Tile has. This must not be changed.
     */
    public final static int AMOUNT_OF_AXIAL_COORDINATES = 2;

    /**
     * How many Cube-Coordinates a Tile has. This must not be changed.
     */
    public final static int AMOUNT_OF_CUBE_COORDINATES = 3;

    /**
     * How many FloatCube-Coordinates a Tile has. This must not be changed.
     */
    public final static int AMOUNT_OF_FLOAT_CUBE_COORDINATES = 3;

    /**
     *
     */
    public final static int COORDINATE_0 = 0;

    /**
     *
     */
    public final static int COORDINATE_1 = 1;

    /**
     *
```

```
     */
    public final static int COORDINATE_2 = 2;

    /**
     *
     */
    public final static int DIVIDE_BY_TWO = 2;

    /**
     *
     */
    public final static int SHORT_DEFAULT = 512;

    /**
     * A Tile's default priority in pathfinding tasks.
     */
    public final static int DEFAULT_PRIORITY = 0;

    /**
     * A Tile's default movement point cost. Currently no other movement point
     * cost is used but the option exists.
     */
    public final static int DEFAULT_MOVEMENT_POINT_COST = 1;

    /**
     * Factor by which the MapConstants.DIAMETER is increased on larger map
     * sizes.
     */
    public static final float[] SIZE_MODIFIERS =
      {
        1f, 1.6f, 2.2f
      };

    /*
     * ------------------------end-ints-------------------------------------
     */
/*
     * -----------------------Start-AxialTransforms---------------------------
     */
    /**
     * The Directions of the AxialTransform. Adding each of these to any one
     * AxialTransform once will return all adjacent spaces.
     */
    /**
     * Bot-Right.
     */
    public static final AxialTransform AXIAL_DIRECTION_0 = new AxialTransform(1, 0);

    /**
     * Bot-Left.
     */
    public static final AxialTransform AXIAL_DIRECTION_1 = new AxialTransform(1, -1);

    /**
     * Left
     */
    public static final AxialTransform AXIAL_DIRECTION_2 = new AxialTransform(0, -1);

    /**
     * Top-Left
     */
    public static final AxialTransform AXIAL_DIRECTION_3 = new AxialTransform(-1, 0);

    /**
     * Top-Right
     */
```

```
    public static final AxialTransform AXIAL_DIRECTION_4 = new AxialTransform(-1, 1);

    /**
     * Right.
     */
    public static final AxialTransform AXIAL_DIRECTION_5 = new AxialTransform(0, 1);

    /**
     * 0=Bot-Right. 1=Bot-Left. 2=Left. 3=Top-Left. 4=Top-Right. 5=Right.
     */
    public static final AxialTransform[] AXIAL_DIRECTIONS =
      {
        AXIAL_DIRECTION_0, AXIAL_DIRECTION_1, AXIAL_DIRECTION_2,
          AXIAL_DIRECTION_3, AXIAL_DIRECTION_4, AXIAL_DIRECTION_5,
      };

    /**
     *
     */
    public static final int TO_HASH = 1000;

    /*
     * -----------------------End---AxialTransforms---------------------------
     */
}
```

6.36.6 MenuConstants

Listing 144: MenuConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 *
 */
public interface MenuConstants
{
    /*---------------------Start-Strings-------------------------------------*/

    /**
     *
     */
    public static final String SELECT_WAND_MULTI = "Select A Wand";

    /**
     *
     */
    public static final String LOBBY_READY = "lobby ready";

    /**
     *
     */
    public static final String SELECT_WAND = "Select A Wand for Player ";

    /**
     *
     */
    public static final String START_GAME = "START\nGAME";

    /**
     *
     */
    public static final String START_LOCAL = "START LOCAL";

    /**
```

```
37      *
38      */
39     public static final String OPTIONS = "OPTIONS";

41     /**
42      *
43      */
44     public static final String END_TURN = "END TURN";

46     /**
47      *
48      */
49     public static final String ENDTURN = "endturn";

51     /**
52      *
53      */
54     public static final String START_NETWORK = "START NETWORK";

56     /**
57      *
58      */
59     public static final String EXIT = "EXIT";

61     /**
62      *
63      */
64     public static final String READY = "READY";

66     /**
67      *
68      */
69     public static final String HOST = "HOST";

71     /**
72      *
73      */
74     public static final String RESUME = "RESUME";

76     /**
77      *
78      */
79     public static final String SAVE_MAP_FILE = "SAVE MAP FILE";

81     /**
82      *
83      */
84     public static final String SOUND = "SOUND";

86     /**
87      *
88      */
89     public static final String YOUR_LOCAL_IP =  "Your Local IP: ";

91     /**
92      *
93      */
94     public static final String ENTER_HOST_IP_HERE = "Enter Host IP here";

96     /**
97      *
98      */
99     public static final String GAME_PORT = ". Game open on port :";

101    /**
102     *
```

```
103         */
104        public static final String MUSIC_ON = "Music_On";

106        /**
107         *
108         */
109        public static final String MUSIC_OFF = "Music_Off";

111        /**
112         *
113         */
114        public static final String VERDANA = "verdana";

116        /**
117         *
118         */
119        public static final String YOUR_GLOBAL_IP = "  Your Global IP: ";

121        /**
122         *
123         */
124        public static final String MAP_OPTIONS = "MAP OPTIONS";

126        /**
127         *
128         */
129        public static final String JOIN_GAME = "Join Game!";

131        /**
132         *
133         */
134        public static final String JOIN = "JOIN";

136        /**
137         *
138         */
139        public static final String BACK = "BACK";

141        /**
142         *
143         */
144        public static final String BACK_TO_MAIN_MENU = "BACK TO MAIN MENU";

146        /**
147         *
148         */
149        public static final String MAP_SHAPE = "MapShape";

151        /**
152         *
153         */
154        public static final String HEXAGON = "Hexagon";

156        /**
157         *
158         */
159        public static final String HEXAGONAL = "Hexagonal";

161        /**
162         *
163         */
164        public static final String RHOMBUS = "Rhombus";

166        /**
167         *
168         */
```

```
169     public static final String MAP_SIZE = "MapSize";

171     /**
172      *
173      */
174     public static final String SMALL = "Small";

176     /**
177      *
178      */
179     public static final String MEDIUM = "Medium";

181     /**
182      *
183      */
184     public static final String LARGE = "Large";

186     /**
187      *
188      */
189     public static final String MAP_RANDOM = "MapRandom";

191     /**
192      *
193      */
194     public static final String RANDOM = "Random";

196     /**
197      *
198      */
199     public static final String NOT_RANDOM = "Not Random";

201     /**
202      *
203      */
204     public static final String SHAPE = "Shape";

206     /**
207      *
208      */
209     public static final String UNKNOWN_HOST = "UnknownHost";

211     /**
212      *
213      */
214     public static final String CHECK_IP = "http://checkip.amazonaws.com";

216     /**
217      *
218      */
219     public static final String LOAD_SAVED_MAP_INFO = "Or load saved "
220             + "map by file name:";

222     /**
223      *
224      */
225     public static final String PRESS_ENTER_INFO = "Press Enter to use this"
226             + " file name\n note that file names are caps-sensitive";

228     /*---------------------End---Strings--------------------------------------*/
229 }
```

6.36.7 NetworkConstants

Listing 145: NetworkConstants.java

```java
package Resource.Constants.GeneralConstants;

/**
 * These are all Constants relevant for the classes in Control.Network
 */
public interface NetworkConstants
{
    /*------------------------Start-ints-------------------------------------*/

    /**
     * The Default port our Servers are listening on.
     */
    public static final int DEFAULT_PORT = 6066;

    /**
     *
     */
    public static final int SEND_THREAD_SLEEP_TIME = 1000;

    /**
     *
     */
    public static final int PACKET_SIZE = 1024;

    /*--------------------------End-ints-------------------------------------*/
    /*------------------------Start-Strings-----------------------------------*/
    /**
     * All Characters that are not in this list are being filtered out of a
     * packet in our PacketDecoder.
     */
    public static final String USEFUL_CHARS = "0123456789qwertzuiopasdfghjkl" +
            "yxcvbnmQWERTZUIOPASDFGHJKLYXCVBNM,.;:-_ ";

    /**
     *
     */
    public static final String GAME_SERVER_THREAD_NAME = "GameServer";

    /**
     *
     */
    public static final String PLAYER_NUMBER_FOR_PACKET = "playernumber ";

    /**
     *
     */
    public static final String LOBBY_FULL = "lobby full";

    /**
     *
     */
    public static final String RECIEVE_THREAD_NAME = "RecieveThread ";

    /**
     *
     */
    public static final String EMPTY = "";

    /**
     *
     */
    public static final String HAND_MINUS_ONE_MINUS = "hand -1 -";

    /**
     *
```

```
     */
    public static final String HAND_MINUS_ONE = "hand -1 ";

    /**
     *
     */
    public static final String HAND = "hand ";

    /**
     *
     */
    public static final String LOBBY_STARTGAME = "lobby startgame";

    /**
     *
     */
    public static final String ABORTING = "Packet invalid. Aborting...";

    /**
     *
     */
    public static final String PACKET_IS_INVALID = "Packet is Invalid.";

    /**
     *
     */
    public static final String SEND_THREAD_NAME = "SendThread ";

    /**
     *
     */
    public static final String USER_THREAD_NAME = "UserThread ";

    /*--------------------------End-String-------------------------------------*/

}
```

6.36.8 PlayerConstants

Listing 146: PlayerConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 *
 */
public interface PlayerConstants
{
    /**
     * Player default value for attack
     */
    public static final int PLAYER_DEFAULT_ATTACK = 15;

    /**
     * The Player default value for mana amount per turn
     */
    public static final int PLAYER_DEFAULT_MANA_AMOUNT = 75;

    /**
     * The Player default value for health
     */
    public static final int PLAYER_DEFAULT_HEALTH = 50;

    /**
     * The Player default value for defense
     */
```

```
    public static final int PLAYER_DEFAULT_DEFENSE = 5;

    /**
     * The Player default value for size
     */
    public static final int PLAYER_DEFAULT_SIZE = 2;

    /**
     * The Player default value for size
     */
    public static final int PLAYER_0 = 0;

    /**
     * The Player default value for size
     */
    public static final int PLAYER_1 = 1;

    /**
     * The Player maximums hand size
     */
    public static final int PLAYER_MAXIMUM_HAND_SIZE = 10;

    /**
     * The Player default value for hand size
     */
    public static final int PLAYER_DEFAULT_STARTING_HAND_SIZE = 8;

    /**
     * The Player default value for movement points
     */
    public static final int PLAYER_DEFAULT_MOVEMENT = 2;
}
```

6.36.9 PopUpConstants

Listing 147: PopUpConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 *
 */
public interface PopUpConstants
{
    /*---------------------Start-Strings-------------------------------------*/

    /**
     *
     */
    public static final String CONNECTION_LOST = "Connection Lost";

    /**
     *
     */
    public static final String CONNECTION_HAS_BEEN_LOST = "Connection has been lost...";

    /**
     *
     */
    public static final String OK = "Ok";

    /**
     *
     */
    public static final String SAVING_MAP = "Saving Map";
```

```
    /**
     *
     */
    public static final String NAME_FOR_MAP = "Enter Name for the .map file";

    /**
     *
     */
    public static final String SAVE = "Save";

    /**
     *
     */
    public static final String SAMPLE_MAP_NAME = "SampleMapName";

    /**
     *
     */
    public static final String GAME_ENDED = "Game Ended";

    /**
     *
     */
    public static final String WINNER_PART_1 = "<<<<<<<<------>>>>>>>>\n\nPlayer ";

    /**
     *
     */
    public static final String WINNER_PART_2 = " Wins!!!\n\n<<<<<<<<------>>>>>>>>";

    /*----------------------End-Strings----------------------------------------*/

}
```

6.36.10 SoundConstants

Listing 148: SoundConstants.java

```
package Resource.Constants.GeneralConstants;

import java.io.File;

/**
 *
 */
public interface SoundConstants {

  /*-------------------------Start-Strings------------------------------*/
    /**
     *
     */
    public static final String MENU_MUSIC = "Art" + File.separator
            + "MenuMusic.mp3";

    /**
     *
     */
    public static final String MAIN_MUSIC = "Art" + File.separator
            + "Main_Music_Theme.mp3";

    /**
     *
     */
    public static final String ADDCHAOS_ADDLAW_MUSIC = "Art" + File.separator
            + "AddChaos_AddLaw_Music.mp3";
```

```
    /**
     *
     */
    public static final String START_NEW_TURN_MUSIC = "Art" + File.separator
            + "StartNewTurn_Music.mp3";

    /**
     *
     */
    public static final String SPAWN_EFFECT = "Art" + File.separator
            + "SpawnEffect.mp3";

    /**
     *
     */
    public static final String[] SOUND_BG = {
        MENU_MUSIC, MAIN_MUSIC, ADDCHAOS_ADDLAW_MUSIC, START_NEW_TURN_MUSIC
    };
  /*-------------------------End-Strings-----------------------------*/
}
```

6.36.11 UIConstants

Listing 149: UIConstants.java

```
package Resource.Constants.GeneralConstants;

/**
 * These are all Constants relevant for initializing our UIElements.
 */
public interface UIConstants
{

    /**
     * Turn-Player-Display x-Location.
     */
    public static final int TURN_PLAYER_DISPLAY_X = 150;

    /**
     * Turn-Player-Display y-Location.
     */
    public static final int TURN_PLAYER_DISPLAY_Y = 100;

    /**
     *
     */
    public static final int FLUX_BAR_Y = 20;

    /**
     * CSS-Stylesheet that is applied to all Sliders in Menus.
     */
    public static final String MENU_SLIDER_STYLE = "view/Stylesheets/MenuSlider.css";

    /**
     * CSS-Stylesheet that is applied to all TextFields in Menus.
     */
    public static final String MENU_TEXT_FIELD_STYLE = "view/Stylesheets/MenuTextField.css";

    /**
     * CSS-Stylesheet that is applied to all MenuButtons ("Drop-Downs") in
     * Menus.
     */
    public static final String MENU_BUTTON_STYLE = "view/Stylesheets/MenuButton.css";

    /**
```

```
     * CSS-Stylesheet that is applied to all MenuItems (items in "Drop-Downs")
     * in Menus.
     */
    public static final String MENU_ITEM_STYLE = "view/Stylesheets/MenuItem.css";

}
```

6.37 Resource.Constants.MinionConstants

6.37.1 PlayerCharacterMinionConstants

Listing 150: PlayerCharacterMinionConstants.java

```
package Resource.Constants.MinionConstants;

/**
 *
 */
public interface PlayerCharacterMinionConstants
{
    /*--------------------------Start-ints-------------------------------------*/
    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_ATTACK_VALUE = 30;

    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_MAGIC_RES_VALUE = 80;

    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_MAX_MOVEMENTPOINTS_VALUE = 2;

    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_MAX_HEALTH_VALUE = 120;

    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_PHYS_RES_VALUE = 10;

    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_SIZE_VALUE = 2;

    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_MOVEMENTPOINTS_VALUE = 2;

    /**
     *
     */
    public static final int DEFAULT_PLAYER_CHARACTER_HEALTH_VALUE = 120;

    /*--------------------------end-ints---------------------------------------*/
    /*--------------------------Start-String-----------------------------------*/

    /**
```

```
     *
     */

    public static final String DEFAULT_PLAYER_CHARACTER_NAME = "Wizard";

    /*-------------------------End-Strings-----------------------------------*/
}
```

6.38 Resource.Constants.MinionConstants.Chaos

6.38.1 HellhoundMinionConstants

Listing 151: HellhoundMinionConstants.java

```
package Resource.Constants.MinionConstants.Chaos;

/**
 *
 */
public interface HellhoundMinionConstants
{
    /*
     * ------------------------------Start-ints-------------------------------
     */

    /**
     *
     */
    public static final int HELLHOUND_MINION_MAX_MOVEMENTPOINTS = 3;

    /**
     *
     */
    public static final int HELLHOUND_MINION_SIZE = 1;

    /**
     *
     */
    public static final int HELLHOUND_MINION_ATTACK = 20;

    /**
     *
     */
    public static final int HELLHOUND_MINION_MAX_HEALTH = 40;

    /**
     *
     */
    public static final int HELLHOUND_MINION_PHYS_RES = 10;

    /**
     *
     */
    public static final int HELLHOUND_MINION_MAGIC_RES = 40;

    /**
     *
     */
    public static final int HELLHOUND_MINION_ATTACK_RANGE = 2;

    /**
     *
     */
    public static final int HELLHOUND_MINION_RANGED_ATTACK = 30;
```

```
    /*
     * ------------------------------End---ints--------------------------------
     */
 /*
     * ------------------------------Start-Strings-----------------------------
     */
    /**
     *
     */
    public static final String HELLHOUND_MINION_NAME = "Hellhound";

    /*
     * ------------------------------End---Strings-----------------------------
     */
}
```

6.38.2 HydraMinionConstants

Listing 152: HydraMinionConstants.java

```
package Resource.Constants.MinionConstants.Chaos;

/**
 *
 */
public interface HydraMinionConstants
{

    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int HYDRA_MINION_MAX_MOVEMENTPOINTS = 2;

    /**
     *
     */
    public static final int HYDRA_MINION_SIZE = 3;

    /**
     *
     */
    public static final int HYDRA_MINION_ATTACK = 70;

    /**
     *
     */
    public static final int HYDRA_MINION_MAX_HEALTH = 280;

    /**
     *
     */
    public static final int HYDRA_MINION_PHYS_RES = 20;

    /**
     *
     */
    public static final int HYDRA_MINION_MAGIC_RES = 120;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/
```

```
    /**
     *
     */


    public static final String HYDRA_MINION_NAME ="Hydra";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.38.3 ManticoreMinionConstants

Listing 153: ManticoreMinionConstants.java

```
package Resource.Constants.MinionConstants.Chaos;

/**
 *
 */
public interface ManticoreMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int MANTICORE_MINION_MAX_MOVEMENTPOINTS = 3;

    /**
     *
     */
    public static final int MANTICORE_MINION_SIZE = 2;

    /**
     *
     */
    public static final int MANTICORE_MINION_ATTACK = 40;

    /**
     *
     */
    public static final int MANTICORE_MINION_MAX_HEALTH = 40;

    /**
     *
     */
    public static final int MANTICORE_MINION_PHYS_RES = 15;

    /**
     *
     */
    public static final int MANTICORE_MINION_MAGIC_RES = 60;

    /**
     *
     */
    public static final int MANTICORE_MINION_ATTACK_RANGE = 3;

    /**
     *
     */
    public static final int MANTICORE_MINION_RANGED_ATTACK = 10;

    /*------------------------------End---ints--------------------------------*/
```

```
    /*------------------------------Start-Strings------------------------------*/

    /**
     *
     */


    public static final String MANTICORE_NAME ="Manticore";

    /*------------------------------End---Strings------------------------------*/
}
```

6.38.4 SkeletonMinionConstants

Listing 154: SkeletonMinionConstants.java

```
package Resource.Constants.MinionConstants.Chaos;

/**
 *
 */
public interface SkeletonMinionConstants
{
    /*------------------------------Start-ints---------------------------------*/

    /**
     *
     */


    public static final int SKELETON_MINION_MAX_MOVEMENTPOINTS = 2;

    /**
     *
     */
    public static final int SKELETON_MINION_SIZE = 1;

    /**
     *
     */
    public static final int SKELETON_MINION_ATTACK = 30;

    /**
     *
     */
    public static final int SKELETON_MINION_MAX_HEALTH = 10;

    /**
     *
     */
    public static final int SKELETON_MINION_PHYS_RES = 5;

    /**
     *
     */
    public static final int SKELETON_MINION_MAGIC_RES = 50;

    /*------------------------------End---ints---------------------------------*/
    /*------------------------------Start-Strings------------------------------*/

    /**
     *
     */


    public static final String SKELETON_NAME ="Skeleton";
```

```
    /*------------------------------End---Strings-----------------------------*/
}
```

6.38.5 VampireMinionConstants

Listing 155: VampireMinionConstants.java

```
package Resource.Constants.MinionConstants.Chaos;

/**
 *
 */
public interface VampireMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int VAMPIRE_MINION_MAX_MOVEMENTPOINTS = 3;

    /**
     *
     */
    public static final int VAMPIRE_MINION_SIZE = 1;

    /**
     *
     */
    public static final int VAMPIRE_MINION_ATTACK = 30;

    /**
     *
     */
    public static final int VAMPIRE_MINION_MAX_HEALTH = 40;

    /**
     *
     */
    public static final int VAMPIRE_MINION_PHYS_RES = 5;

    /**
     *
     */
    public static final int VAMPIRE_MINION_MAGIC_RES = 60;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String VAMPIRE_NAME ="Vampire";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.38.6 ZombieMinionConstants

Listing 156: ZombieMinionConstants.java

```
package Resource.Constants.MinionConstants.Chaos;

/**
 *
 */
public interface ZombieMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int ZOMBIE_MINION_MAX_MOVEMENTPOINTS = 1;

    /**
     *
     */
    public static final int ZOMBIE_MINION_SIZE = 2;

    /**
     *
     */
    public static final int ZOMBIE_MINION_ATTACK = 50;

    /**
     *
     */
    public static final int ZOMBIE_MINION_MAX_HEALTH = 30;

    /**
     *
     */
    public static final int ZOMBIE_MINION_PHYS_RES = 5;

    /**
     *
     */
    public static final int ZOMBIE_MINION_MAGIC_RES = 20;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String ZOMBIE_NAME ="Zombie";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.39 Resource.Constants.MinionConstants.Law

6.39.1 IcarusTowerMinionConstants

Listing 157: IcarusTowerMinionConstants.java

```
package Resource.Constants.MinionConstants.Law;

/**
```

```
 *
 */
public interface IcarusTowerMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int ICARUS_TOWER_MINION_MAX_MOVEMENTPOINTS = 1;

    /**
     *
     */
    public static final int ICARUS_TOWER_MINION_SIZE = 2;

    /**
     *
     */
    public static final int ICARUS_TOWER_MINION_ATTACK = 10;

    /**
     *
     */
    public static final int ICARUS_TOWER_MINION_MAX_HEALTH = 50;

    /**
     *
     */
    public static final int ICARUS_TOWER_MINION_PHYS_RES = 10;

    /**
     *
     */
    public static final int ICARUS_TOWER_MINION_MAGIC_RES = 0;

    /**
     *
     */
    public static final int ICARUS_TOWER_MINION_ATTACK_RANGE = 3;

    /**
     *
     */
    public static final int ICARUS_TOWER_MINION_RANGED_ATTACK = 40;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String ICARUS_TOWER_NAME ="Icarus Tower";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.39.2 PaladinMinionConstants

Listing 158: PaladinMinionConstants.java

```
package Resource.Constants.MinionConstants.Law;
```

```
/**
 *
 */
public interface PaladinMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int PALADIN_MINION_MAX_MOVEMENTPOINTS = 2;

    /**
     *
     */
    public static final int PALADIN_MINION_SIZE = 2;

    /**
     *
     */
    public static final int PALADIN_MINION_ATTACK = 50;

    /**
     *
     */
    public static final int PALADIN_MINION_MAX_HEALTH = 80;

    /**
     *
     */
    public static final int PALADIN_MINION_PHYS_RES = 20;

    /**
     *
     */
    public static final int PALADIN_MINION_MAGIC_RES = 100;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String PALADIN_NAME ="Paladin";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.39.3 PegasusMinionConstants

Listing 159: PegasusMinionConstants.java

```
package Resource.Constants.MinionConstants.Law;

/**
 *
 */
public interface PegasusMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/
```

```
    /**
     *
     */


    public static final int PEGASUS_MINION_MAX_MOVEMENTPOINTS = 4;

    /**
     *
     */
    public static final int PEGASUS_MINION_SIZE = 1;

    /**
     *
     */
    public static final int PEGASUS_MINION_ATTACK = 10;

    /**
     *
     */
    public static final int PEGASUS_MINION_MAX_HEALTH = 30;

    /**
     *
     */
    public static final int PEGASUS_MINION_PHYS_RES = 5;

    /**
     *
     */
    public static final int PEGASUS_MINION_MAGIC_RES = 90;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String PEGASUS_NAME ="Pegasus";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.39.4 SapphireDragonMinionConstants

Listing 160: SapphireDragonMinionConstants.java

```
package Resource.Constants.MinionConstants.Law;

/**
 *
 */
public interface SapphireDragonMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int SAPPHIRE_DRAGON_MINION_MAX_MOVEMENTPOINTS = 3;

    /**
```

```
     *
     */
    public static final int SAPPHIRE_DRAGON_MINION_SIZE = 2;

    /**
     *
     */
    public static final int SAPPHIRE_DRAGON_MINION_ATTACK = 70 ;

    /**
     *
     */
    public static final int SAPPHIRE_DRAGON_MINION_MAX_HEALTH = 110 ;

    /**
     *
     */
    public static final int SAPPHIRE_DRAGON_MINION_PHYS_RES = 10 ;

    /**
     *
     */
    public static final int SAPPHIRE_DRAGON_MINION_MAGIC_RES = 140;

    /**
     *
     */
    public static final int SAPPHIRE_DRAGON_MINION_ATTACK_RANGE = 3;

    /**
     *
     */
    public static final int SAPPHIRE_DRAGON_MINION_RANGED_ATTACK = 40;

    /*------------------------------End---ints-------------------------------*/
    /*------------------------------Start-Strings----------------------------*/

    /**
     *
     */


    public static final String SAPPHIRE_DRAGON_NAME ="Sapphire Dragon";

    /*------------------------------End---Strings----------------------------*/
}
```

6.40 Resource.Constants.MinionConstants.Neutral

6.40.1 AirElementalMinionConstants

Listing 161: AirElementalMinionConstants.java

```
package Resource.Constants.MinionConstants.Neutral;

/**
 *
 */
public interface AirElementalMinionConstants
{
    /*------------------------------Start-ints-------------------------------*/

    /**
     *
     */
```

```
    public static final int AIR_ELEMENTAL_MINION_MAX_MOVEMENTPOINTS = 3;

    /**
     *
     */
    public static final int AIR_ELEMENTAL_MINION_SIZE = 1;

    /**
     *
     */
    public static final int AIR_ELEMENTAL_MINION_ATTACK = 80;

    /**
     *
     */
    public static final int AIR_ELEMENTAL_MINION_MAX_HEALTH = 60;

    /**
     *
     */
    public static final int AIR_ELEMENTAL_MINION_PHYS_RES = 20;

    /**
     *
     */
    public static final int AIR_ELEMENTAL_MINION_MAGIC_RES = 70;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String AIR_ELEMENTAL_NAME ="Air Elemental";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.40.2 EagleMinionConstants

Listing 162: EagleMinionConstants.java

```
package Resource.Constants.MinionConstants.Neutral;

/**
 *
 */
public interface EagleMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int EAGLE_MINION_MAX_MOVEMENTPOINTS = 4;

    /**
     *
     */
    public static final int EAGLE_MINION_SIZE = 1;
```

```
    /**
     *
     */
    public static final int EAGLE_MINION_ATTACK = 10;

    /**
     *
     */
    public static final int EAGLE_MINION_MAX_HEALTH = 30;

    /**
     *
     */
    public static final int EAGLE_MINION_PHYS_RES = 5;

    /**
     *
     */
    public static final int EAGLE_MINION_MAGIC_RES = 10;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String EAGLE_NAME ="Eagle";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.40.3 GiantMinionConstants

Listing 163: GiantMinionConstants.java

```
package Resource.Constants.MinionConstants.Neutral;

/**
 *
 */
public interface GiantMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int GIANT_MINION_MAX_MOVEMENTPOINTS = 2;

    /**
     *
     */
    public static final int GIANT_MINION_SIZE = 3;

    /**
     *
     */
    public static final int GIANT_MINION_ATTACK = 120;

    /**
     *
```

```
     */
    public static final int GIANT_MINION_MAX_HEALTH = 100;

    /**
     *
     */
    public static final int GIANT_MINION_PHYS_RES = 25;

    /**
     *
     */
    public static final int GIANT_MINION_MAGIC_RES = 70;

    /**
     *
     */
    public static final int GIANT_MINION_ATTACK_RANGE = 3;

    /**
     *
     */
    public static final int GIANT_MINION_RANGED_ATTACK = 80;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */
    public static final String GIANT_NAME ="Giant";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.40.4 LionMinionConstants

Listing 164: LionMinionConstants.java

```
package Resource.Constants.MinionConstants.Neutral;

/**
 *
 */
public interface LionMinionConstants
{
    /*------------------------------Start-ints--------------------------------*/

    /**
     *
     */


    public static final int LION_MINION_MAX_MOVEMENTPOINTS = 3;

    /**
     *
     */
    public static final int LION_MINION_SIZE = 1;

    /**
     *
     */
    public static final int LION_MINION_ATTACK = 40;

    /**
     *
```

```
     */
    public static final int LION_MINION_MAX_HEALTH = 40;

    /**
     *
     */
    public static final int LION_MINION_PHYS_RES = 10;

    /**
     *
     */
    public static final int LION_MINION_MAGIC_RES = 20;

    /*------------------------------End---ints--------------------------------*/
    /*------------------------------Start-Strings-----------------------------*/

    /**
     *
     */


    public static final String LION_NAME ="Lion";

    /*------------------------------End---Strings-----------------------------*/
}
```

6.41 Resource.Images

6.41.1 Animations

Listing 165: Animations.java

```
package Resource.Images;

import java.io.File;

/**
 * All FilePaths to our Animations
 *
 */
public interface Animations
{

    /**
     * This Animation is played when a creature takes Damage.
     */
    public static final String HIT = "Art" + File.separator + "Hit.gif";

    /**
     * This Animation is played when a creature is hit by a spell.
     */
    public static final String SPELLHIT = "Art" + File.separator
            + "BlueExplosion.gif";

    /**
     * This Animation is played when a creature dies.
     */
    public static final String DEATH = "Art" + File.separator + "Death.gif";

    /**
     * This Animation is played when a creature is buffed.
     */
    public static final String BUFF = "Art" + File.separator + "Buff.gif";

    /**
```

```
     * This Animation is played when a creature is summoned.
     */
    public static final String SUMMON = "Art" + File.separator + "Summon.gif";
}
```

6.41.2 CardExclusiveImages

Listing 166: CardExclusiveImages.java

```
package Resource.Images;

import java.io.File;

/**
 * The paths to the Images that are exclusive used on cards are stored in this Interface.
 */
public interface CardExclusiveImages
{

    /**
     * Path to the image of Decree.
     */
    public static final String DECREE = "Art" + File.separator
            + "Decree.png";

    /**
     * Path to the image of Divine Bow.
     */
    public static final String DIVINE_BOW = "Art" + File.separator
            + "DivineBow.png";

    /**
     * Path to the image of Divine Shield.
     */
    public static final String DIVINE_SHIELD = "Art" + File.separator
            + "DivineShield.png";

    /**
     * Path to the image of Divine Sword.
     */
    public static final String DIVINE_SWORD = "Art" + File.separator
            + "DivineSword.png";

    /**
     * Path to the image of Magic Bolt.
     */
    public static final String MAGIC_BOLT = "Art" + File.separator
            + "MagicBolt.png";

    /**
     * Path to the image of Sapphire Dragon Super.
     */
    public static final String SAPPHIRE_DRAGON_SUPER = "Art" + File.separator
            + "SapphireDragonSuper.png";

    /**
     * Path to the image of Vengeance.
     */
    public static final String VENGEANCE = "Art" + File.separator
            + "Vengeance.png";

    /**
     * Path to the image of Windwalker.
     */
    public static final String WINDWALKER = "Art" + File.separator
            + "Windwalker.png";
```

```
}
```

6.41.3 GOImages

Listing 167: GOImages.java

```
package Resource.Images;

import java.io.File;

/**
 * The paths to the images of all GameObjects are stored in this interface.
 */
public interface GOImages
{
    /**
     * Path to the image of Air Elemental.
     */
    public static final String AIR_ELEMENTAL = "Art" + File.separator
            + "AirElemental.png";

    /**
     * Path to the image of Eagle.
     */
    public static final String EAGLE = "Art" + File.separator
            + "Eagle.png";

    /**
     * Path to the image of Giant.
     */
    public static final String GIANT = "Art" + File.separator
            + "Giant.png";

    /**
     * Path to the image of Hellhound.
     */
    public static final String HELLHOUND = "Art" + File.separator
            + "Hellhound.png";

    /**
     * Path to the image of Hydra.
     */
    public static final String HYDRA = "Art" + File.separator
            + "Hydra.png";

    /**
     * Path to the image of Icarus Tower.
     */
    public static final String ICARUS_TOWER = "Art" + File.separator
            + "IcarusTower.png";

    /**
     * Path to the image of Lion.
     */
    public static final String LION = "Art" + File.separator
            + "SummonableCreature_Lion.png";

    /**
     * Path to the image of Mana Crystal.
     */
    public static final String MANA_CRYSTAL = "Art" + File.separator
            + "Mana_Crystal.png";

    /**
     * Path to the image of Manticore.
```

```
60          */
61         public static final String MANTICORE = "Art" + File.separator
62                 + "Manticore.png";

64         /**
65          * Path to the image of Obstacle.
66          */
67         public static final String OBSTACLE = "Art" + File.separator
68                 + "Obstacle.png";

70         /**
71          * Path to the image of Paladin.
72          */
73         public static final String PALADIN = "Art" + File.separator
74                 + "Paladin.png";

76         /**
77          * Path to the image of Pegasus.
78          */
79         public static final String PEGASUS = "Art" + File.separator
80                 + "Pegasus.png";

82         /**
83          * Path to the image of the green Wizard.
84          */
85         public static final String PLAYER0_CHARACTER = "Art" + File.separator
86                 + "PlayerCharacter_Wizard_red.png";

88         /**
89          * Path to the image of the green Wizard Frame.
90          */
91         public static final String PLAYER0_FRAME = "Art" + File.separator
92                 + "Frame_Player_Red.png";

94         /**
95          * Path to the image of the yellow Wizard.
96          */
97         public static final String PLAYER1_CHARACTER = "Art" + File.separator
98                 + "PlayerCharacter_Wizard_yellow.png";

100        /**
101         * Path to the image of the yellow Wizard Frame.
102         */
103        public static final String PLAYER1_FRAME = "Art" + File.separator
104                + "Frame_Player_Yellow.png";

106        /**
107         * Path to the image of the Sapphire Dragon.
108         */
109        public static final String SAPPHIRE_DRAGON = "Art" + File.separator
110                + "SapphireDragon.png";

112        /**
113         * Path to the image of Skeleton.
114         */
115        public static final String SKELETON = "Art" + File.separator
116                + "Skeleton.png";

118        /**
119         * Path to the image of Vampire.
120         */
121        public static final String VAMPIRE = "Art" + File.separator
122                + "Vampire.png";

124        /**
125         * Path to the image of Zombie.
```

```
     */
    public static final String ZOMBIE = "Art" + File.separator
            + "Zombie.png";

}
```

6.41.4 HexImages

Listing 168: HexImages.java

```
package Resource.Images;

import java.io.File;

/**
 * The paths to the images that are related to hex are stored in this interface.
 */
public interface HexImages
{
    /**
     * Path to the image of basic Hex Frame.
     */
    public static final String HEX_FRAME = "Art" + File.separator
            + "Hex_Frame.png";

    /**
     * Path to the image of Hex Frame highlited for attack.
     */
    public static final String HEX_FRAME_HIGHLIGHTED_ATTACK = "Art"
            + File.separator + "Hex_Frame_Highlighted_atk.png";

    /**
     * Path to the image of Hex Frame highlited for move.
     */
    public static final String HEX_FRAME_HIGHLIGHTED_MOVE = "Art"
            + File.separator + "Hex_Frame_Highlighted_move.png";

    /**
     * Path to the image of Hex Frame highlited for selection.
     */
    public static final String HEX_FRAME_SELECTED = "Art"
            + File.separator + "Hex_Frame_Selected.png";

    /**
     * String Array with the paths to the images for different hex heights,
     * index in array is equal to hex height.
     */
    public static final String[] HEX_HEIGHTS =
      {
        "Art" + File.separator + "Hex_Height_0.png", "Art" + File.separator +
         "Hex_Height_1.png", "Art" + File.separator + "Hex_Height_2.png", "Art"
          + File.separator + "Hex_Height_3.png"
      };

}
```

6.41.5 Images

Listing 169: Images.java

```
package Resource.Images;

import java.io.File;

/**
```

```
 * The Main Imageinterface that stores some basic backgrounds and also serves
 * all images in one interface with inheriting from all other imageinterfaces.
 */
public interface Images extends HexImages, GOImages, CardExclusiveImages,
        WandImages, UIElementImages
{
    /**
     * Path to the image of the Menu background 0.
     */
    public static final String MENU0 = "Art" + File.separator
            + "MenuBackground0.png";

    /**
     * Path to the image of the menu background 1.
     */
    public static final String MENU1 = "Art" + File.separator
            + "MenuBackground1.png";

    /**
     * Path to the image of the menu background 2.
     */
    public static final String MENU2 = "Art" + File.separator
            + "MenuBackground2.png";

    /**
     * Path to the image of the menu background 3.
     */
    public static final String MENU3 = "Art" + File.separator
            + "MenuBackground3.png";

    /**
     * Path to the image of the game background.
     */
    public static final String GAME_BG = "Art" + File.separator
            + "GameBackground.jpg";

    /**
     * Path to the image of for a case where is no image.
     */
    public static final String NO_IMAGE = "Art" + File.separator
            + "NoImage.png";

    /**
     * String Array with the paths to Menu backgrounds, 0 to 3.
     */
    public static final String[] MENU_BG =
      {
        MENU0, MENU1, MENU2, MENU3
      };

}
```

6.41.6 UIElementsImages

Listing 170: UIElementsImages.java

```
package Resource.Images;

import java.io.File;

/**
 * The paths to the images of the User Interface Elements are stored in this
 * interface.
 */
public interface UIElementImages
{
```

```
    /**
     * Path to the image of the deck.
     */
    public static final String DECK = "Art" + File.separator + "deck.png";

    /**
     * Path to the image of the end turn hover.
     */
    public static final String END_TURN_HOVER = "Art" + File.separator
            + "EndTurnHover.png";

    /**
     * Path to the image of the end turn idle.
     */
    public static final String END_TURN_IDLE = "Art" + File.separator
            + "EndTurnIdle.png";

    /**
     * Path to the image of the end turn pressed.
     */
    public static final String END_TURN_PRESSED = "Art" + File.separator
            + "EndTurnPressed.png";

    /**
     * Path to the image of the flux bar.
     */
    public static final String FLUX_BAR = "Art" + File.separator
            + "Flux.png";

    /**
     * Path to the image of the mana bar.
     */
    public static final String MANA_BAR = "Art" + File.separator
            + "ManaBar.png";

    /**
     * Path to the image of the mana cost.
     */
    public static final String MANA_COST = "Art" + File.separator
            + "ManaCost.png";
}
```

6.41.7 WandImages

Listing 171: WandImages.java

```
package Resource.Images;

import java.io.File;

/**
 * Paths to the Wand images are stored in this interface.
 */
public interface WandImages
{

    /**
     * Path to the image of the Air Wand.
     */
    public static final String AIR_WAND = "Art" + File.separator
            + "AirWand.png";

    /**
     * Path to the image of the Creatue Wand.
     */
```

```
    public static final String CREATURE_WAND = "Art" + File.separator
            + "CreatureWand.png";

    /**
     * Path to the image of the Undead Wand.
     */
    public static final String UNDEAD_WAND = "Art" + File.separator
            + "UndeadWand.png";

    /**
     * Path to the image of the Random Wand.
     */
    public static final String RANDOM_WAND = "Art" + File.separator
            + "RandomWand.png";
}
```

6.42 Util

6.42.1 Stack

Listing 172: Stack.java

```
package Util;

import java.util.ArrayList;
import java.util.List;

/**
 * Implementation of a minimal Stack.
 */
public class Stack
{

    /**
     * Object List, used for the Stack implementation.
     */
    private List<Object> _stack = null;

    /**
     * Default Constructor.
     */
    public Stack ()
    {
        this._stack = new ArrayList();
    }

    /**
     * Removes a Object from the top of the stack and returns it.
     *
     * @return The Object on top of the stack.
     */
    public Object pop ()
    {
        int length = this._stack.size();

        Object top = this._stack.get(length - 1);

        this._stack.remove(length - 1);

        return top;
    }

    /**
     * Adds a new Object to the top of the Stack.
     *
```

```
     * @param o The new Object.
     */
    public void put (Object o)
    {
        this._stack.add(o);
    }

    /**
     * Determines the size of the stack.
     *
     * @return The length of the stack.
     */
    public int size ()
    {
        return this._stack.size();
    }

}
```

6.43 View.Menus

6.43.1 MainMenu

Listing 173: MainMenu.java

```
package View.Menus;

import Control.Controllers.GameController;
import Control.Controllers.SceneController;
import Control.Controllers.SoundController;
import static Resource.Constants.GeneralConstants.MenuConstants.*;
import Resource.Images.Images;
import View.Scenes.ChaosScene;
import View.Scenes.GameSetupScene;
import View.UIElements.ChaosBackground;
import View.UIElements.ChaosMenuButton;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 */
public class MainMenu extends Menu
{

    /**
     * Creates a new Main Menu.
     *
     * @param scene Scene this menu is part of.
     */
    public MainMenu (ChaosScene scene)
    {
        super(scene);
        SoundController.getInstance().getMusic().playMenuMusic();

    }

    private ChaosMenuButton createLocalButton ()
    {
        ChaosMenuButton btnStart = new ChaosMenuButton(START_LOCAL);
        btnStart.setOnMouseClicked(event ->
          {
            GameController.getInstance().initGame(false);
            SceneController.getInstance().setScene(new GameSetupScene());
```

```
            GameController.getInstance().startGame();
          });
        return btnStart;
    }

    private ChaosMenuButton createOptionsButton ()
    {
        ChaosMenuButton btnOptions = new ChaosMenuButton(OPTIONS);
        btnOptions.setOnMouseClicked(event ->
          {
            nextMenu(new OptionsMenu(this.getChaosScene()));
          });
        return btnOptions;
    }

    private ChaosMenuButton createMPButton ()
    {
        ChaosMenuButton btnNetwork = new ChaosMenuButton(START_NETWORK);
        btnNetwork.setOnMouseClicked(event ->
          {
            nextMenu(new MultiplayerMenu(this.getChaosScene()));
          });
        return btnNetwork;
    }

    private ChaosMenuButton createExitButton ()
    {
        ChaosMenuButton btnExit = new ChaosMenuButton(EXIT);
        btnExit.setOnMouseClicked(event ->
          {
            System.exit(0);
          });
        return btnExit;
    }

    /**
     * Creates the Main Menu's buttons and implements their functionality.
     */
    @Override
    public void initialize ()
    {
        super.initialize();
        ChaosMenuButton btnStart = this.createLocalButton();
        ChaosMenuButton btnNetwork = this.createMPButton();
        ChaosMenuButton btnOptions = this.createOptionsButton();
        ChaosMenuButton btnExit = this.createExitButton();

        this.getMenuElements().getChildren().addAll(btnStart, btnNetwork,
                btnOptions, btnExit);
    }

    /**
     * Gets the background image from the resources package.
     */
    @Override
    public void initBackground ()
    {
        try
          {
            this.setBg(new ChaosBackground(Images.MENU0));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(MainMenu.class.getName()).log(Level.SEVERE, null, ex);
          }
    }
```

```
}
```

6.43.2 MapOptionsMenu

Listing 174: MapOptionsMenu.java

```
package View.Menus;

import Model.Map.MapGenArgs;
import Model.Map.MapGenerationKeywords;
import static Resource.Constants.GeneralConstants.GameObjectConstants.SIZE;
import static Resource.Constants.GeneralConstants.MenuConstants.*;
import static Resource.Constants.GeneralConstants.NetworkConstants.EMPTY;
import Resource.Constants.GeneralConstants.UIConstants;
import Resource.Images.Images;
import View.Scenes.ChaosScene;
import View.UIElements.ChaosBackground;
import View.UIElements.ChaosMenuButton;
import View.UIElements.MapView;
import View.UIElements.TextWithBackground;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.control.Label;
import javafx.scene.control.MenuButton;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.
        ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.RowConstraints;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;

/**
 *
 */
public class MapOptionsMenu extends Menu
{

    private MapView mapView = null;

    /**
     * Creates a new Game Options Menu
     *
     * @param scene Scene this menu is part of.
     */
    public MapOptionsMenu (ChaosScene scene)
    {
        super(scene);
    }

    private MenuButton createShapeMenuButton ()
    {
        MenuButton mapShapeMB = new MenuButton(MAP_SHAPE);

        MenuItem hexagonMI = new MenuItem(HEXAGON);
        hexagonMI.setOnAction(event ->
          {
            MapGenArgs.setShape(MapGenerationKeywords.hexagon);
            mapShapeMB.setText(RHOMBUS);
          });
```

```
        MenuItem rhombusMI = new MenuItem(RHOMBUS);
        rhombusMI.setOnAction(event ->
          {
            MapGenArgs.setShape(MapGenerationKeywords.rhombus);
            mapShapeMB.setText(RHOMBUS);
          });
        mapShapeMB.getItems().addAll(hexagonMI, rhombusMI);
        return mapShapeMB;
    }

    private MenuButton createSizeMenuButton ()
    {
        MenuButton mapSizeMB = new MenuButton(MAP_SIZE);

        MenuItem smallMI = new MenuItem(SMALL);
        smallMI.setOnAction(event ->
          {
            MapGenArgs.setSize(MapGenerationKeywords.small);
            mapSizeMB.setText(SMALL);
          });

        MenuItem mediumMI = new MenuItem(MEDIUM);
        mediumMI.setOnAction(event ->
          {
            MapGenArgs.setSize(MapGenerationKeywords.medium);
            mapSizeMB.setText(MEDIUM);
          });

        MenuItem largeMI = new MenuItem(LARGE);
        largeMI.setOnAction(event ->
          {
            MapGenArgs.setSize(MapGenerationKeywords.large);
            mapSizeMB.setText(LARGE);
          });

        mapSizeMB.getItems().addAll(smallMI, mediumMI, largeMI);
        return mapSizeMB;
    }

    private MenuButton createRandomMenuButton ()
    {
        MenuButton mapRandomMB = new MenuButton(MAP_RANDOM);

        MenuItem randomMI = new MenuItem(RANDOM);

        randomMI.setOnAction(event ->
          {
            MapGenArgs.setRandom(MapGenerationKeywords.random);
            mapRandomMB.setText(RANDOM);
          });

        MenuItem noRandomMI = new MenuItem(NOT_RANDOM);
        noRandomMI.setOnAction(event ->
          {
            MapGenArgs.setRandom(MapGenerationKeywords.noRandom);
            mapRandomMB.setText(NOT_RANDOM);
          });

        mapRandomMB.getItems().addAll(randomMI, noRandomMI);
        return mapRandomMB;
    }

    private ChaosMenuButton createBackButton ()
    {
        ChaosMenuButton btnBack = new ChaosMenuButton(BACK);
        btnBack.setOnMouseClicked(event ->
```

```
            {
              back(new OptionsMenu(this.getChaosScene()));
            });
        return btnBack;
    }

    private TextField createFileNameTextField ()
    {
        TextField fileNameTF = new TextField();
        fileNameTF.setOnMouseClicked(evt ->
            {
              fileNameTF.setText(EMPTY);
            });
        fileNameTF.setOnKeyPressed(evt ->
            {
              if (evt.getCode() == KeyCode.ENTER)
                {
                  MapGenArgs.setFileName(fileNameTF.getText());
                }
            }
        );
        return fileNameTF;
    }

    /**
     * Creates the Game Options Menu's buttons and implements their
     * functionality.
     */
    @Override
    public void initialize ()
    {
        super.initialize();
        this.getMenuElements().setTranslateX(_offset);

        //grid pane to lay out items
        GridPane grid = new GridPane();
        for (int i = 0; i < 3; i++)
            {
              grid.getColumnConstraints().add(new ColumnConstraints(100));
              grid.getRowConstraints().add(new RowConstraints(32));
            }

        //apply stylesheets
        this.getStylesheets().add(UIConstants.MENU_BUTTON_STYLE);
        this.getStylesheets().add(UIConstants.MENU_ITEM_STYLE);

        TextWithBackground shapeText = new TextWithBackground(SHAPE, 20);
        grid.add(shapeText, 0, 0);

        MenuButton mapShapeMB = this.createShapeMenuButton();
        grid.add(mapShapeMB, 0, 1);

        TextWithBackground sizeText = new TextWithBackground(SIZE , 20);
        grid.add(sizeText, 1, 0);

        MenuButton mapSizeMB = this.createSizeMenuButton();
        grid.add(mapSizeMB, 1, 1);

        TextWithBackground randomText = new TextWithBackground(RANDOM, 20);
        grid.add(randomText, 2, 0);

        MenuButton mapRandomMB = this.createRandomMenuButton();
        grid.add(mapRandomMB, 2, 1);

        ChaosMenuButton btnBack = this.createBackButton();
```

```
        //load file by name
        this.getStylesheets().add(UIConstants.MENU_TEXT_FIELD_STYLE);

        Label fileNameLabel1 = new Label(LOAD_SAVED_MAP_INFO);
        fileNameLabel1.setFont(new Font(16));
        fileNameLabel1.setTextFill(Color.WHITE);
        TextField fileNameTF = this.createFileNameTextField();

        Label fileNameLabel2 = new Label(PRESS_ENTER_INFO);
        fileNameLabel2.setFont(new Font(16));
        fileNameLabel2.setTextFill(Color.WHITE);

        Rectangle spaceholder1 = new Rectangle(30, 30);
        spaceholder1.setOpacity(0d);
        Rectangle spaceholder2 = new Rectangle(30, 30);
        spaceholder2.setOpacity(0d);

        this.getMenuElements().getChildren().addAll(grid, spaceholder1, btnBack,
                spaceholder2, fileNameLabel1, fileNameTF, fileNameLabel2);
    }

    /**
     * Gets the background image from the resources package.
     */
    @Override
    public void initBackground ()
    {
        try
          {
            this.setBg(new ChaosBackground(Images.MENU3));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(MainMenu.class.getName()).log(Level.SEVERE, null, ex);
          }
    }

}
```

6.43.3 Menu

Listing 175: Menu.java

```
package View.Menus;

import View.Scenes.ChaosScene;
import View.UIElements.ChaosBackground;
import javafx.animation.TranslateTransition;
import javafx.scene.Parent;
import javafx.scene.layout.VBox;
import javafx.util.Duration;

/**
 *
 */
public abstract class Menu extends Parent
{

    /**
     * The scene this menu is part of.
     */
    private ChaosScene _scene = null;

    /**
     * The width of slide-effect when moving between menus.
     */
```

```
    final int _offset;

    /**
     * This menus background.
     */
    private ChaosBackground _bg = null;

    /**
     * Contains this menus Buttons to lay them out under each other.
     */
    private VBox _menuElements = null;

    /**
     * Constructs a new menu and initializes menu elements.
     *
     * @param scene Scene this menu is part of.
     */
    public Menu (ChaosScene scene)
    {
        this._offset = 400;
        this._menuElements = new VBox(10);
        this._scene = scene;
        this.initBackground();
        this.initialize();
    }

    /**
     * initializes the menu background and buttons.
     */
    public void initialize ()
    {
        if (_bg != null)
          {
            _bg.scaleToMonitorSize();
            this.getChildren().add(this._bg);
          }
        this._menuElements.setLayoutX(100);
        this._menuElements.setLayoutY(200);
        this.getChildren().add(this._menuElements);
    }

    /**
     * Gets the background image from the resources package.
     */
    public void initBackground ()
    {
        //to be overridden
    }

    /**
     * Switches to next menu.
     *
     * @param next menu to be displayed
     */
    public void nextMenu (Menu next)
    {
        _scene.addNodes(next);

        TranslateTransition tt0 = new TranslateTransition(Duration.seconds(0.25),
                this.getMenuElements());
        tt0.setToX(this.getMenuElements().getTranslateX() - _offset);

        TranslateTransition tt1 = new TranslateTransition(Duration.seconds(0.5),
                next.getMenuElements());
        tt1.setToX(this.getMenuElements().getTranslateX());
```

```
        tt0.setOnFinished(event ->
          {
            _scene.removeNode(this);
          });
        tt0.play();
        tt1.play();
    }

    /**
     * Switches to last menu. Identical to nextMenu but slide-effect direction
     * is changed.
     *
     * @param next The last menu.
     */
    public void back (Menu next)
    {
        next.getMenuElements().setTranslateX(-_offset);
        _scene.addNodes(next);

        TranslateTransition tt0 = new TranslateTransition(Duration.seconds(0.25),
                this.getMenuElements());
        tt0.setToX(this.getMenuElements().getTranslateX() + _offset);

        TranslateTransition tt1 = new TranslateTransition(Duration.seconds(0.5),
                next.getMenuElements());
        tt1.setToX(this.getMenuElements().getTranslateX());

        tt0.setOnFinished(event ->
          {
            _scene.removeNode(this);
          });
        tt0.play();
        tt1.play();
    }

    /**
     *
     * @return the menus background.
     */
    public ChaosBackground getBg ()
    {
        return _bg;
    }

    /**
     * Sets this menus background.
     *
     * @param background The desired Background
     */
    public void setBg (ChaosBackground background)
    {
        _bg = background;
    }

    /**
     *
     * @return this menus menuElements-VBox.
     */
    public VBox getMenuElements ()
    {
        return _menuElements;
    }

    /**
     *
     * @return the scene this menu is part of.
```

```
     */
    public ChaosScene getChaosScene ()
    {
        return _scene;
    }

}
```

6.43.4 MultiplayerMenu

Listing 176: MultiplayerMenu.java

```
package View.Menus;

import Control.Controllers.GameController;
import Control.Network.GameClient;
import Control.Network.GameServer;
import static Resource.Constants.GeneralConstants.MenuConstants.*;
import Resource.Constants.GeneralConstants.NetworkConstants;
import static Resource.Constants.GeneralConstants.NetworkConstants.EMPTY;
import Resource.Constants.GeneralConstants.UIConstants;
import Resource.Images.Images;
import View.Scenes.ChaosScene;
import View.UIElements.ChaosBackground;
import View.UIElements.ChaosMenuButton;
import View.UIElements.TextWithBackground;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.InputStreamReader;
import java.net.InetAddress;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;

/**
 *
 */
public class MultiplayerMenu extends Menu
{

    public MultiplayerMenu (ChaosScene scene)
    {
        super(scene);
    }

    private void joinGame (TextField ipTF)
    {
        GameClient client = new GameClient(ipTF.getText(), NetworkConstants.
                DEFAULT_PORT);
        client.setPlayerNumber(1);
        GameController.getInstance().setClient(client);
        client.execute();
    }

    private ChaosMenuButton createHostButton ()
    {
        ChaosMenuButton btnHost = new ChaosMenuButton(HOST);
        btnHost.setOnMouseClicked(event ->
          {
            GameController.getInstance().initGame(true);
            String ipText = UNKNOWN_HOST;

            String globalIP = EMPTY;
            String localIP = EMPTY;
```

```
            URL whatismyip;
            try
              {
                whatismyip = new URL(CHECK_IP);
                BufferedReader in = new BufferedReader(new InputStreamReader(
                        whatismyip.openStream()));
                globalIP = in.readLine();
                localIP = InetAddress.getLocalHost().getHostAddress();

              }
            catch (Exception ex)
              {
                Logger.getLogger(MultiplayerMenu.class.getName()).log(Level.
                        SEVERE, null, ex);
              }

            ipText =YOUR_LOCAL_IP + localIP + YOUR_GLOBAL_IP +
                    globalIP + GAME_PORT +
                    NetworkConstants.DEFAULT_PORT;

            TextWithBackground twb = new TextWithBackground(ipText, 20);
            twb.setTranslateY(15);
            this.getMenuElements().getChildren().add(twb);

            //host
            GameServer server = new GameServer(NetworkConstants.DEFAULT_PORT);
            server.start();

            //join
            GameClient client;

            client = new GameClient(globalIP, NetworkConstants.DEFAULT_PORT);
            client.setPlayerNumber(0);
            GameController.getInstance().setClient(client);
            client.execute();
          });
        return btnHost;
    }

    private ChaosMenuButton createJoinButton ()
    {
        ChaosMenuButton btnJoin = new ChaosMenuButton(JOIN);
        btnJoin.setOnMouseClicked(event ->
          {

            GameController.getInstance().initGame(true);

            TextField ipTF = new TextField();
            ipTF.setText(ENTER_HOST_IP_HERE);
            ipTF.setRotate(-2);
            ipTF.setOnMouseClicked(evt ->
              {
                ipTF.setText(EMPTY);
              });
            ipTF.setOnKeyPressed(evt ->
              {
                if (evt.getCode() == KeyCode.ENTER)
                  {
                    this.joinGame(ipTF);
                  }
              }
            );
            this.getMenuElements().getChildren().add(ipTF);

            ChaosMenuButton btnConfirmJoin = new ChaosMenuButton(JOIN_GAME);
```

```
            btnConfirmJoin.setOnMouseClicked(evt ->
              {
                this.joinGame(ipTF);
              }
            );
            this.getMenuElements().getChildren().add(btnConfirmJoin);

          });
        return btnJoin;
    }

    private ChaosMenuButton createBackButton ()
    {
        ChaosMenuButton btnBack = new ChaosMenuButton(BACK);
        btnBack.setOnMouseClicked(event ->
          {
            GameController.getInstance().shutDown();

            back(new MainMenu(this.getChaosScene()));
          });
        return btnBack;
    }

    @Override
    public void initialize ()
    {
        super.initialize();
        this.getStylesheets().add(UIConstants.MENU_TEXT_FIELD_STYLE);

        this.getMenuElements().setTranslateX(_offset);

        ChaosMenuButton btnHost = this.createHostButton();
        ChaosMenuButton btnJoin = this.createJoinButton();
        ChaosMenuButton btnBack = this.createBackButton();

        this.getMenuElements().getChildren().addAll(btnHost, btnJoin, btnBack);
    }

    @Override
    public void initBackground ()
    {
        try
          {
            this.setBg(new ChaosBackground(Images.MENU3));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(OptionsMenu.class.getName()).log(Level.SEVERE,
                    null, ex);
          }
    }

}
```

6.43.5 OptionsMenu

Listing 177: OptionsMenu.java

```
package View.Menus;

import Control.Controllers.SceneController;
import static Resource.Constants.GeneralConstants.MenuConstants.*;
import Resource.Images.Images;
import View.Scenes.ChaosScene;
import View.UIElements.ChaosBackground;
import View.UIElements.ChaosMenuButton;
```

```
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 */
public class OptionsMenu extends Menu
{

    /**
     * Creates a new Options Menu
     *
     * @param scene Scene this menu is part of.
     */
    public OptionsMenu (ChaosScene scene)
    {
        super(scene);
    }

    private ChaosMenuButton createMapOptionsButton ()
    {
        ChaosMenuButton btnMap = new ChaosMenuButton(MAP_OPTIONS);
        btnMap.setOnMouseClicked(event ->
          {
            nextMenu(new MapOptionsMenu(this.getChaosScene()));
          });
        return btnMap;
    }

    private ChaosMenuButton createSoundOptionsButton ()
    {
        ChaosMenuButton btnSound = new ChaosMenuButton(SOUND);
        btnSound.setOnMouseClicked(event ->
          {
            nextMenu(new SoundMenu(this.getChaosScene()));
          });
        return btnSound;
    }

    private ChaosMenuButton createBackButton ()
    {
        ChaosMenuButton btnBack = new ChaosMenuButton(BACK);
        btnBack.setOnMouseClicked(event ->
          {
            if (SceneController.getInstance().isIsPaused())
              {
                this.getChaosScene().removeNode(this);
                this.getChaosScene().addNodes(new PauseMenu(this.
                        getChaosScene()));
              }
            else
              {
                back(new MainMenu(this.getChaosScene()));
              }
          });
        return btnBack;
    }

    /**
     * Creates the Options Menu's buttons and implements their functionality.
     */
    @Override
    public void initialize ()
    {
        super.initialize();
```

```
        this.getMenuElements().setTranslateX(_offset);

        ChaosMenuButton btnMap = this.createMapOptionsButton();
        ChaosMenuButton btnSound = this.createSoundOptionsButton();
        ChaosMenuButton btnBack = this.createBackButton();

        this.getMenuElements().getChildren().addAll(btnMap, btnSound, btnBack);
    }

    /**
     * Gets the background image from the resources package.
     */
    @Override
    public void initBackground ()
    {
        try
          {
            this.setBg(new ChaosBackground(Images.MENU1));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(OptionsMenu.class.getName()).log(Level.SEVERE,
                    null, ex);
          }
    }

}
```

6.43.6 PauseMenu

Listing 178: PauseMenu.java

```
package View.Menus;

import Control.Controllers.GameController;
import Control.Controllers.SceneController;
import Control.Controllers.SoundController;
import static Resource.Constants.GeneralConstants.MenuConstants.*;
import Resource.Dimensions;
import View.PopUps.MapSavePopUp;
import View.Scenes.ChaosScene;
import View.Scenes.MenuScene;
import View.UIElements.ChaosMenuButton;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;

/**
 *
 */
public class PauseMenu extends Menu
{

    /**
     * Creates a new Pause Menu.
     *
     * @param scene Scene this menu is part of.
     */
    public PauseMenu (ChaosScene scene)
    {
        super(scene);
    }

    private ChaosMenuButton createOptionsButton ()
    {
        ChaosMenuButton btnOptions = new ChaosMenuButton(OPTIONS);
```

```
        btnOptions.setOnMouseClicked(event ->
          {
            nextMenu(new OptionsMenu(this.getChaosScene()));
            SoundController.getInstance().getMusic().pauseGameMusic();
            SoundController.getInstance().getMusic().playMenuMusic();
          });
        return btnOptions;
    }

    private ChaosMenuButton createSaveMapButton ()
    {
        ChaosMenuButton btnSaveMap = new ChaosMenuButton(SAVE_MAP_FILE);
        btnSaveMap.setOnMouseClicked(event ->
          {
            new MapSavePopUp().display();
          });
        return btnSaveMap;
    }

    private ChaosMenuButton createResumeButton ()
    {
        ChaosMenuButton btnResume = new ChaosMenuButton(RESUME);
        btnResume.setOnMouseClicked(event ->
          {
            SceneController.getInstance().setIsPaused(false);
            this.getChaosScene().removeNode(this);
            SoundController.getInstance().getMusic().stopMenuMusic();
            SoundController.getInstance().getMusic().playGameMusic();
          });
        return btnResume;
    }

    private ChaosMenuButton createMainMenuButton ()
    {
        ChaosMenuButton btnMenu = new ChaosMenuButton(BACK_TO_MAIN_MENU);
        btnMenu.setOnMouseClicked(event ->
          {
            SceneController.getInstance().setIsPaused(false);
            GameController.getInstance().shutDown();
            SceneController.getInstance().setScene(new MenuScene());
          });
        return btnMenu;
    }

    /**
     * Creates the Pause Menu's buttons and implements their functionality.
     */
    @Override
    public void initialize ()
    {
        super.initialize();

        ChaosMenuButton btnOptions = this.createOptionsButton();
        ChaosMenuButton btnSaveMap = this.createSaveMapButton();
        ChaosMenuButton btnResume = this.createResumeButton();
        ChaosMenuButton btnMenu = this.createMainMenuButton();

        this.getMenuElements().getChildren().addAll(btnOptions, btnSaveMap,
                btnResume, btnMenu);
    }

    /**
     * Background for the pause menu is the currently running game, partly
     * covered by a gray rectangle, which is created here.
     */
    @Override
```

```
    public void initBackground ()
    {
        Rectangle bg = new Rectangle();
        bg.setHeight(Dimensions.getHeight());
        bg.setWidth(Dimensions.getWidth());
        bg.setFill(Color.DARKGRAY);
        bg.setOpacity(0.3);
        bg.setMouseTransparent(false);
        this.getChildren().add(bg);
    }

}
```

6.43.7 SoundMenu

Listing 179: SoundMenu.java

```
package View.Menus;

import Control.Controllers.SceneController;
import Control.Controllers.SoundController;
import static Resource.Constants.GeneralConstants.MenuConstants.*;
import Resource.Constants.GeneralConstants.UIConstants;
import Resource.Images.Images;
import View.Scenes.ChaosScene;
import View.UIElements.ChaosBackground;
import View.UIElements.ChaosMenuButton;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Orientation;
import javafx.scene.control.Label;
import javafx.scene.control.RadioButton;
import javafx.scene.control.Slider;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;

/**
 *
 */
public class SoundMenu extends Menu
{

    /**
     * Creates a new Sound Menu.
     *
     * @param scene Scene this menu is part of.
     */
    public SoundMenu (ChaosScene scene)
    {
        super(scene);
    }

    private ChaosMenuButton createBackButton ()
    {
        ChaosMenuButton btnBack = new ChaosMenuButton(BACK);
        btnBack.setOnMouseClicked(event ->
          {
            back(new OptionsMenu(this.getChaosScene()));
          });
        return btnBack;
    }
```

```
    private Slider createSoundSlider ()
    {
        SceneController.getInstance().getCurrentScene().getStylesheets().
                add(UIConstants.MENU_SLIDER_STYLE);

        Slider musicVolume = new Slider(0, 80, 60);
        musicVolume.setOrientation(Orientation.HORIZONTAL);
        musicVolume.valueProperty().addListener(new ChangeListener()
        {

            @Override
            public void changed (ObservableValue observable, Object oldValue,
                    Object newValue)
            {
                SoundController.getInstance().getMusic().
                        setVolume(musicVolume.getValue() / 80);
            }

        });

        return musicVolume;
    }

    private GridPane createRadioButtonGridPane ()
    {
        GridPane radioButtons = new GridPane();

        //labels for the RadioButtons
        Label labelOn = new Label(MUSIC_ON);
        Label labelOff = new Label(MUSIC_OFF);
        labelOn.setFont(Font.font(VERDANA, 20));
        labelOn.setTextFill(Color.DARKGRAY);
        labelOff.setFont(Font.font(VERDANA, 20));
        labelOff.setTextFill(Color.DARKGRAY);
        radioButtons.add(labelOn, 0, 0);
        radioButtons.add(labelOff, 0, 1);

        //creation of the RadioButtons
        ToggleGroup tGroup = new ToggleGroup();
        RadioButton music_On = new RadioButton();
        RadioButton music_Off = new RadioButton();
        music_On.setToggleGroup(tGroup);
        music_Off.setToggleGroup(tGroup);
        music_On.setFocusTraversable(false);
        music_Off.setFocusTraversable(false);
        if (SoundController.getInstance().getMusic().isMusic())
          {
            music_On.setSelected(true);
          }
        else
          {
            music_Off.setSelected(true);
          }
        //adding RadioButtons
        radioButtons.add(music_On, 1, 0);
        radioButtons.add(music_Off, 1, 1);
        radioButtons.setHgap(25);

        //ChangeListener for enabeling or disabeling sound
        tGroup.selectedToggleProperty().addListener(new ChangeListener()
        {
            @Override
            public void changed (ObservableValue observable, Object oldValue,
                    Object newValue)
            {
```

```
                    if (newValue.equals(music_On))
                      {
                        SoundController.getInstance().getMusic().setMusic(true);
                        SoundController.getInstance().getMusic().playMenuMusic();
                      }
                    else
                      {
                        SoundController.getInstance().getMusic().stopMenuMusic();
                        SoundController.getInstance().getMusic().setMusic(false);
                      }
                }

            });
            return radioButtons;
        }

        /**
         * Creates the Sound Menus UI-Elements and implements their functionality.
         */
        @Override
        public void initialize ()
        {
            GridPane grid = new GridPane();

            /*
             * sets layout coordinates to move the menu from the top-left corner
             * toward the screen center
             */
            super.initialize();

            /*
             * offset is used for transitions between menus
             */
            this.getMenuElements().setTranslateX(_offset);

            ChaosMenuButton btnBack = this.createBackButton();
            GridPane radioButtons = this.createRadioButtonGridPane();
            Slider musicVolume = this.createSoundSlider();

            grid.add(radioButtons, 0, 0);
            grid.add(musicVolume, 0, 1);

            this.getMenuElements().getChildren().addAll(grid, btnBack);
        }

        /**
         * Gets the background image from the resources package.
         */
        @Override
        public void initBackground ()
        {
            try
              {
                this.setBg(new ChaosBackground(Images.MENU0));
              }
            catch (FileNotFoundException ex)
              {
                Logger.getLogger(SoundMenu.class.getName()).log(Level.SEVERE, null,
                        ex);
              }
        }

    }
```

6.44 View.PopUps

6.44.1 ConnectionPopUp

Listing 180: ConnectionPopUp.java

```
package View.PopUps;

import Control.Controllers.GameController;
import Control.Controllers.SceneController;
import static Resource.Constants.GeneralConstants.PopUpConstants.*;
import View.Scenes.MenuScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

/**
 * The ConnectionPopUp pops up on the Screen if the Computer lost the connection
 * to the other PLayer while the Game.
 */
public class ConnectionPopUp extends PopUp
{

    /**
     * The ConnectionPopUp pops up on the Screen if the Computer lost the
     * connection to the other PLayer while the Game.
     */
    public ConnectionPopUp ()
    {
        super();
        GameController.getInstance().shutDown();
    }

    /**
     * Displays the ConnectionPopUp on the Screen.
     */
    @Override
    public void display ()
    {
        this.setWindowTitle(CONNECTION_LOST);

        Label label1 = new Label(CONNECTION_HAS_BEEN_LOST);

        Button btnOk = new Button(OK);

        btnOk.setOnAction(e ->
          {
            SceneController.getInstance().setScene(new MenuScene());

            this.closeWindow();
          });

        this.addToWindow(label1, btnOk);

        this.showAndWait();

    }

}
```

6.44.2 MapSavePopUp

Listing 181: MapSavePopUp.java

```
package View.PopUps;
```

```
import Model.Map.MapGenerator;
import static Resource.Constants.GeneralConstants.PopUpConstants.*;
import Resource.Constants.GeneralConstants.UIConstants;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;

/**
 * The MapSavePopUp appears on the Screen if you want to save a .map file.
 */
public class MapSavePopUp extends PopUp
{

    /**
     * Displays the MapSavePopUp on the Screen.
     */
    @Override
    public void display ()
    {
        this.setWindowTitle(SAVING_MAP);

        Label label1 = new Label(NAME_FOR_MAP);

        Button btnSave = new Button(SAVE);

        TextField fileNameTF = new TextField();
        fileNameTF.setText(SAMPLE_MAP_NAME);
        fileNameTF.getStylesheets().add(UIConstants.MENU_TEXT_FIELD_STYLE);

        btnSave.setOnAction(e ->
          {
            MapGenerator.saveMap(fileNameTF.getText());

            this.closeWindow();
          });

        this.addToWindow(label1, fileNameTF, btnSave);

        this.showAndWait();

    }

}
```

6.44.3 PopUp

Listing 182: PopUp.java

```
package View.PopUps;

import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import javafx.stage.Modality;
import javafx.stage.Stage;

/**
 * PopUp is the super class for all inheriting PopUps. It provides a VBox and a
 * own Stage.
 */
public abstract class PopUp
{
```

```
    /**
     * The Layout of the PopUp.
     */
    private VBox _layout = null;

    /**
     * The Stage of this PopUp open a new Window.
     */
    private Stage _popupwindow = null;

    /**
     * PopUp is the super class for all inheriting PopUps.
     * It provides a VBox and a
     * own Stage.
     */
    public PopUp ()
    {
        this._layout = new VBox(10);
        this._popupwindow = new Stage();
        _popupwindow.initModality(Modality.APPLICATION_MODAL);

        _layout.setAlignment(Pos.CENTER);

        Scene scene1 = new Scene(_layout, 300, 250);

        _popupwindow.setScene(scene1);

    }

    /**
     * Sets the Title of the PopUp.
     *
     * @param title - the title.
     */
    public void setWindowTitle (String title)
    {
        this._popupwindow.setTitle(title);
    }

    /**
     * Closes the PopUp.
     */
    public void closeWindow ()
    {
        this._popupwindow.close();
    }

    /**
     * Adds different elements to the PopUp.
     *
     * @param label1 - add label
     * @param fileNameTF - add textfield
     * @param btnSave - add button
     */
    public void addToWindow (Label label1,TextField fileNameTF,Button btnSave)
    {
        this._layout.getChildren().addAll(label1, fileNameTF, btnSave);
    }

    /**
     * Adds different elements to the PopUp.
     *
     * @param label1 - add label
     * @param btnSave - add button
     */
```

```
    public void addToWindow (Label label1 ,Button btnSave)
    {
        this._layout.getChildren().addAll(label1, btnSave);
    }

    /**
     * Shows the PopUp.
     */
    public void showAndWait ()
    {
        this._popupwindow.showAndWait();
    }

    /**
     * Displays the PopUp, needs to be overwritten in all inheriting classes.
     */
    public abstract void display ();
}
```

6.44.4 WinPopUp

Listing 183: WinPopUp.java

```
package View.PopUps;

import Control.Controllers.GameController;
import Control.Controllers.SceneController;
import static Resource.Constants.GeneralConstants.PopUpConstants.*;
import View.Scenes.MenuScene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

/**
 * The WinPopUp pops up if a Player won the Game and
 * informs everybody about this.
 */
public class WinPopUp extends PopUp
{

    /**
     * Owner Number of the Winner.
     */
    private int _winner = 0;

    /**
     * The WinPopUp pops up if a Player won the Game and
     * informs everybody about this.
     *
     * @param winner - the owner Number of the Winner.
     */
    public WinPopUp (int winner)
    {
        super();
        this._winner = -1;
        GameController.getInstance().shutDown();
        this._winner = winner +1;
    }

    /**
     * Displays the PopUp on the screen.
     */
    @Override
    public void display ()
    {
        this.setWindowTitle(GAME_ENDED);
```

```
        Label label1 = new Label(WINNER_PART_1 + _winner +
            WINNER_PART_2);

        Button btnOk = new Button(OK);

        btnOk.setOnAction(e ->
          {
            GameController.getInstance().shutDown();

            SceneController.getInstance().setScene(new MenuScene());

            this.closeWindow();
          });

        this.addToWindow(label1, btnOk);

        this.showAndWait();
    }

}
```

6.45 View.Scenes

6.45.1 ChaosScene

Listing 184: ChaosScene.java

```
package View.Scenes;

import Control.Controllers.SceneController;
import Resource.Dimensions;
import Resource.VideoOptions;
import View.UIElements.IChaosUI;
import javafx.application.Platform;
import javafx.scene.Group;
import javafx.scene.Node;

/**
 * The ChaosScene is the base Sceneclass that offers the main functions for
 * adding and removing nodes or groups and for scaling the scene to the
 * devices monitorsize.
 */
public abstract class ChaosScene
{
    /**
     * Group that holds all given Nodes
     */
    private Group _sceneGroup = null;

    /**
     * True if scene is already scaled to current monitorsize
     */
    private boolean _scaledToMonitorSize = false;

    /**
     * Default Constructor.
     */
    public ChaosScene ()
    {
        this._sceneGroup = new Group();
    }

    /**
     * Constructs a Scene with an already existing Pack of Nodes.
     *
```

```
     * @param nodeList the Nodes, the scene will be initialized with.
     */
    public ChaosScene (Node[] nodeList)
    {
        this._sceneGroup = new Group();
        this.addNodes(nodeList);
    }

    /**
     * Scales a single UIElement on the highest layer of this scene's children
     * from a default 4K vertical resolution to the Users current actual screen
     * vertical resolution, overridden in most UIElements.
     *
     * @param cui The UIElement to be scaled.
     */
    private static void scaleToHeight (IChaosUI cui)
    {
        // The Users device actual vertical resolution in pixels.
        double monitorHeight = Dimensions.getHeight();

        // The Original 4K vertical resolution in pixels.
        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;

        // The difference between original 4K height and the users monitor height.
        double heightDifference = originalHeight - monitorHeight;

        Node n = (Node) cui;

        //Calculates the factor, with which the UIElements size is to be set to,
        //to fit the new resolution.
        double newScale = n.getScaleY() * (monitorHeight / originalHeight);

        n.setScaleY(newScale);

        //Calculates the offset, which the UIElement has to be moved by, to be
        //in the same relative position where it would be at Default resolution.
        double newLayoutY = (n.getTranslateY() - heightDifference *
            (n.getTranslateY() / originalHeight));

        n.setTranslateY(newLayoutY);

    }

    /**
     * scales a single UIElement on the highest layer of this scene's children
     * from a default 4K horizontal resolution to the Users current actual
     * screen horizontal resolution, overridden in most UIElements.
     *
     * @param cui - The UIElement to be scaled.
     */
    private static void scaleToWidth (IChaosUI cui)
    {
        // The Users actual vertical resolution in pixels.
        double monitorWidth = Dimensions.getWidth();

        // The Original 4K vertical resolution in pixels.
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        // The difference between original 4K width and the users monitor width.
        double widthDifference = originalWidth - monitorWidth;

        Node n = (Node) cui;

        //Calculates the factor, with which the UIElements size is to be set to,
        //to fit the new resolution.
        double newScale = n.getScaleX() * (monitorWidth / originalWidth);
```

```
        n.setScaleX(newScale);

        //Calculates the offset, which the UIElement has to be moved by, to be
        //in the same relative position where it would be at Default resolution.
        double newLayoutX = (n.getTranslateX() - widthDifference *
            (n.getTranslateX() / originalWidth));

        n.setTranslateX(newLayoutX);

    }

    /**
     * Scales a given UIElement from a default 4K resolution to the Users actual
     * screen resolution.
     *
     * @param cui - The UIElement to be scaled.
     */
    public static void scaleToMonitorSize (IChaosUI cui)
    {
        scaleToHeight(cui);
        scaleToWidth(cui);
    }

    /**
     * Scales all of this scene's top layer nodes from a default 4K resolution
     * to the Users actual screen resolution,
     * can only be done once per scene to prevent nodes being scaled multiple
     * times.
     */
    public void scaleAllSceneNodesToMonitorSize ()
    {
        //Terminates if this Scene has already been scaled.
        if (this._scaledToMonitorSize)
          {
            return;
          }

        for (Node n : this._sceneGroup.getChildren())
          {

            if (IChaosUI.class.isAssignableFrom(n.getClass()))
              {
                ((IChaosUI) n).scaleToMonitorSize();
              }

          }

        this._scaledToMonitorSize = true;
    }

    /**
     * Used to refresh all Nodes of this Scene,
     * to be overwritten in all scenes individually.
     *
     * @return - this Scene
     */
    public abstract ChaosScene refresh ();

    /**
     * Adds a Single node to be displayed in this scene.
     *
     * @param n - The Node to be added.
     */
    public void addNodes (Node n)
    {
```

```
171         try
172           {
173             this._sceneGroup.getChildren().add(n);
174           }
175         catch (Exception e)
176           {
177             reportAndLogException(e, new Runnable()
178                           {
179                               @Override
180                               public void run ()
181                               {
182                                   SceneController.getInstance()
183                                       .getCurrentChaosScene()._sceneGroup
184                                       .getChildren().add(n);
185                               }
186                           });
187           }

189     }

191     /**
192      * Reports and Logs an Exception later.
193      *
194      * @param t   - to reported Exception.
195      * @param run - Runnable to be ran later.
196      */
197     public void reportAndLogException (final Throwable t, Runnable run)
198     {
199         Platform.runLater(run);
200     }

202     /**
203      * Adds multiple nodes to be displayed in this scene.
204      *
205      * @param n - An Array of nodes, which are to be added to the scene.
206      */
207     public void addNodes (Node[] n)
208     {
209         for (Node node : n)
210           {
211             this.addNodes(node);
212           }
213     }

215     /**
216      * forcibly sets this scene group as the given Group. All existing nodes in
217      * this scene will thereby be removed.
218      *
219      * @param g - the new Scenegroup
220      */
221     public void setGroup (Group g)
222     {
223         this._sceneGroup = g;
224     }

226     /**
227      * Used to access all Visible Nodes on the highest layer of this scenegroup.
228      *
229      * @return - all Visible Nodes on this scenes highest layer.
230      */
231     public Node[] getNodes ()
232     {
233         Node[] nodes = new Node[this._sceneGroup.getChildren().size()];
234         for (int i = 0; i < nodes.length; i++)
235           {
236             nodes[i] = this._sceneGroup.getChildren().get(i);
```

```
            }
        return nodes;
    }

    /**
     * Used to access all of this scene's nodes.
     *
     * @return - The SceneGroup holding all of this scene's nodes.
     */
    public Group getGroup ()
    {
        return this._sceneGroup;
    }

    /**
     * Removes all Nodes from this scene.
     */
    public void clear ()
    {
        this._sceneGroup.getChildren().clear();
    }

    /**
     * Removes a single given node from this scene.
     *
     * @param n - the node to be removed.
     */
    public void removeNode (Node n)
    {
        this._sceneGroup.getChildren().remove(n);
    }

}
```

6.45.2 GameScene

Listing 185: GameScene.java

```
package View.Scenes;

import Control.Controllers.AnimationController;
import Control.Controllers.GameController;
import Control.Controllers.SceneController;
import Control.Network.Packets.Packet;
import static Resource.Constants.GeneralConstants.MenuConstants.ENDTURN;
import static Resource.Constants.GeneralConstants.MenuConstants.END_TURN;
import Resource.Dimensions;
import Resource.Images.Images;
import View.Menus.PauseMenu;
import View.Tooltip.TooltipBox;
import View.UIElements.*;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.input.KeyCode;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;

/**
 * The GameScene displays the Game with all necessary Elements that are needed
 * to play the game.
 */
public class GameScene extends ChaosScene
{

    /**
```

```
29      * The view on the current map of the current Game.
30      */
31     private MapView _mapView = null;

33     /**
34      * The ManaBar of this Game.
35      */
36     private ManaBar _manaBar = null;

38     /**
39      * The Fluxbar of this Game.
40      */
41     private FluxBar _fluxBar = null;

43     /**
44      * The Button to end the Turn.
45      */
46     private RoundButton _endTurnButton = null;

48     /**
49      * Displays the Game specially for the Player whos turn it is.
50      */
51     private TurnPlayerDisplay _turnPlayerDisplay = null;

53     /**
54      * The View on the Deck of the Current Player in this Game.
55      */
56     private DeckUI _deckUI = null;

58     /**
59      * The View on the Hand of the current Player.
60      */
61     private HandView _handView = null;

63     /**
64      * The Background of the Game.
65      */
66     private ChaosBackground _bg = null;

68     /**
69      * The Rectangle of this Game.
70      */
71     private Rectangle _rec = null;

73     /**
74      * The GameScene displays the Game with all necessary Elements that are
75      * needed
76      * to play the game.
77      */
78     public GameScene ()
79     {
80         this._mapView = new MapView();
81         this._manaBar = new ManaBar();
82         this._fluxBar = new FluxBar();
83         this._endTurnButton = new RoundButton(END_TURN);
84         this._turnPlayerDisplay = new TurnPlayerDisplay();
85         this._deckUI = new DeckUI();
86         this._handView = new HandView();
87         this.init();
88     }

90     /**
91      * Initializes the Attributes of new GameScene Object.
92      */
93     private void init ()
94     {
```

```
        this.initBackground();
        this.initRectangle();
        this.initTooltips();
        this.initMap();
        this.initUIElements();
        this.initButtons();
        this.initKeyPressedEvents();
        this.refresh();
    }

    /**
     * Inits the Game Background.
     */
    private void initBackground ()
    {
        try
          {
            _bg = new ChaosBackground(Images.GAME_BG);
            this.addNodes(this._bg);
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameSetupScene.class.getName()).
                log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Inits the Tooltips of the Game.
     */
    private void initTooltips ()
    {
        this.addNodes(TooltipBox.getBox());
    }

    /**
     * Inits the Map of the Game.
     */
    private void initMap ()
    {
        this.addNodes(this._mapView);
        this._mapView.setScaleX(2);
        this._mapView.setScaleY(2);
        AnimationController.getInstance().setMapView(this._mapView);
    }

    /**
     * Inits the Buttons in the Game.
     */
    private void initButtons ()
    {
        this._endTurnButton.setOnMouseReleased(event ->
          {
            this._endTurnButton.onMouseReleased();

            if (GameController.getInstance().getCurrentGame().isMultiplayer())
              {
                GameController.getInstance().getClient().
                    sendPackage(new Packet(ENDTURN));
              }

            GameController.getInstance().getCurrentTurnController().nextPlayer();
            SceneController.getInstance().refresh();
          });

        this.addNodes(this._endTurnButton);
```

```
    }

    /**
     * Inits the keypressedevents in the GameScene, if you press esc on your
     * keyboard a Pausemenu will appear on the Screen.
     */
    private void initKeyPressedEvents ()
    {
        SceneController.getInstance().getCurrentScene().setOnKeyPressed(event ->
          {
            //Pause Menu on Escape presses
            if ((event.getCode() == KeyCode.ESCAPE) && !SceneController.
                getInstance().isIsPaused())
              {
                SceneController.getInstance().setIsPaused(true);
                displayPauseMenu();
              }
          });
    }

    /**
     * Helping method for initKeyPressedEvents.
     */
    private void displayPauseMenu ()
    {
        this.addNodes(new PauseMenu(this));
    }

    /**
     * Adds all initialized UIElements to the GanmeScene.
     */
    private void initUIElements ()
    {
        this.addNodes(this._turnPlayerDisplay);
        this.addNodes(this._handView);
        this.addNodes(this._fluxBar);
        this.addNodes(this._manaBar);

        this._manaBar.setScaleX(1.5);
        this._manaBar.setScaleY(1.5);

        this.addNodes(this._deckUI);
    }

    /**
     * refreshs the Map to display changes.
     */
    private void refreshMap ()
    {
        this._mapView.refresh();
    }

    /**
     * refreshs other UIElements to display Changes after a Player made some
     * turns.
     */
    private void refreshUIElements ()
    {
        this._turnPlayerDisplay.refresh();
        this._fluxBar.refresh();
        this._manaBar.refresh();
        this._deckUI.refresh();
        this._handView.refresh();

        this.removeNode(_rec);
```

```
        if (GameController.getInstance().getCurrentGame().isMultiplayer() &&
                (GameController.getInstance().getCurrentTurnController()
                    .getCurrentPlayer().getOwnerNumber() !=
                 GameController.getInstance().getClient().getPlayerNumber()))
          {
            this.addNodes(_rec);
          }
    }

    /**
     * Inits the Rectangle.
     */
    private void initRectangle ()
    {
        this._rec = new Rectangle();
        this._rec.setHeight(Dimensions.getHeight());
        this._rec.setWidth(Dimensions.getWidth());
        this._rec.setFill(Color.DARKGRAY);
        this._rec.setOpacity(0.1);
        this._rec.setMouseTransparent(false);
    }

    /**
     * Refreshs the whole Scene
     *
     * @return - a new refreshed ChaosScene.
     */
    @Override
    public ChaosScene refresh ()
    {
        this.refreshUIElements();
        this.refreshMap();

        return this;
    }

}
```

6.45.3 GameSetupScene

Listing 186: GameSetupScene.java

```
package View.Scenes;

import Control.Controllers.GameController;
import Control.Controllers.SceneController;
import Control.Controllers.SoundController;
import Model.Card.Wand;
import Model.Card.WandFactory;
import static Resource.Constants.GeneralConstants.MenuConstants.*;
import Resource.Images.Images;
import Resource.VideoOptions;
import View.Tooltip.TooltipBox;
import View.UIElements.*;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.geometry.Pos;
import javafx.scene.effect.DropShadow;
import javafx.scene.paint.Color;

/**
 * The GameSetupScene displays which Wand each Player has chosen and which Wands
 * are chosable, it also provides a button to start the game.
 */
public class GameSetupScene extends ChaosScene
```

```
{
    /**
     * X-Coordinate of this Scene.
     */
    private int _midX = 0;

    /**
     * Y-Coordinate of this Scene.
     */
    private int _midY = 0;

    /**
     * The most left coordinate of this Scene to Adjust the Wandview.
     */
    private double _leftMost = 0.0;

    /**
     * The Players its turn it is to choose a Wand.
     */
    private int _choosingPlayer = 0;

    /**
     * The Button to start the Game.
     */
    private RoundButton _startButton = null;

    /**
     * The View onto the chosable Wands.
     */
    private WandView[] _wands = null;

    /**
     * VBox where the PlayerWandViews are displayed in.
     */
    private ChaosVBox _pWVBox = null;

    /**
     * PlayerWandViews are stored here.
     */
    private PlayerWandView[] _pWVs = null;

    /**
     * The Background of this Scene.
     */
    private ChaosBackground _bg = null;

    /**
     * Shows Text with an own Background.
     */
    private TextWithBackground _text = null;

    /**
     * The GameSetupScene displays which Wand each Player has chosen and
     * which Wands are chosable, it also provides a button to start the game.
     */
    public GameSetupScene ()
    {
        this._startButton = new RoundButton(START_GAME);
        this._wands = new WandView[WandFactory.getWandCount()];
        this._pWVBox = new ChaosVBox();
        this._pWVs = new PlayerWandView[2];
        this._midX = (int) (VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH
                / 2 - 289 / 2);
        this._midY = (int) (VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT
                / 2) - 250;
```

```
        this._leftMost = _midX - 289 * 1.5d * (_wands.length - 1) / 2 - 11
                * (_wands.length - 1) / 2;
        this.init();
    }

    /**
     * Inits all Attributes of the Scene to display it.
     */
    public void init ()
    {
        this.initBackground();
        this.initMenuButton();
        this.initText();
        this.initWandViews();
        this.initTooltips();
        this.initPlayerWandViews();
        this.initStartButton();
    }

    /**
     * inits the Tooltips.
     */
    private void initTooltips ()
    {
        this.addNodes(TooltipBox.getBox());
    }

    /**
     * Inits the Texts.
     */
    private void initText ()
    {
        this._text = new TextWithBackground(SELECT_WAND
                + this._choosingPlayer, 75);
        this._text.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT
                / 2 - 450);
        this._text.setTranslateX(VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH
                / 2 - this._text.getTotalWidth() * 1.5d / 2);

        this.addNodes(this._text);
    }

    /**
     * Inits the Background of this Scene.
     */
    private void initBackground ()
    {
        try
          {
            _bg = new ChaosBackground(Images.MENU3);
            this.addNodes(this._bg);
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameSetupScene.class.getName()).
                log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Inits the view on to the Wands you can choose.
     */
    private void initWandViews ()
    {
        for (int i = 0; i < this._wands.length; i++)
          {
```

```
            this._wands[i] = new WandView(i);

            this._wands[i].setScaleX(1.5d);
            this._wands[i].setScaleY(1.5d);

            this._wands[i].setTranslateY(_midY);
            this._wands[i].setTranslateX(this._leftMost + i * 289 * 1.5d + 16
                    * (i - 5));

            this._wands[i].setOnMouseClicked(event ->
              {
                this.chooseWand(((WandView) event.getSource()).getWand());
              });

            this.addNodes(this._wands[i]);
          }

    }

    /**
     * Inits the view on to the Wands the Players have chosen.
     */
    private void initPlayerWandViews ()
    {
        this._pWVBox.setAlignment(Pos.CENTER_LEFT);
        this._pWVBox.setTranslateX(80);
        this._pWVBox.setScaleX(1.5d);
        this._pWVBox.setScaleY(1.5d);
        this._pWVBox.setMouseTransparent(true);

        for (int i = 0; i < _pWVs.length; i++)
          {
            this._pWVs[i] = new PlayerWandView(i);

            this._pWVBox.getChildren().add(this._pWVs[i].get());
          }

        this.addNodes(this._pWVBox);
    }

    /**
     * Inits the Button to the menu.
     */
    private void initMenuButton ()
    {
        ChaosMenuButton mb = new ChaosMenuButton(BACK_TO_MAIN_MENU);
        mb.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT - 300);
        mb.setTranslateX(150);
        mb.setScaleX(2);
        mb.setScaleY(2);

        mb.setOnMouseReleased(event ->
          {
            GameController.getInstance().shutDown();
            SceneController.getInstance().setScene(new MenuScene());
          });

        this.addNodes(mb);
    }

    /**
     * Inits the Button you have to press to start the Game.
     */
    private void initStartButton ()
    {
```

```
        this._startButton.setOnMouseReleased(event ->
          {
            this._startButton.onMouseReleased();

            for (PlayerWandView wv : this._pWVs)
              {
                if (!wv.hasWand())
                  {
                    return;
                  }
              }
            SoundController.getInstance().getMusic().stopMenuMusic();
            SoundController.getInstance().getMusic().playStartGame();

            SceneController.getInstance().setScene(new GameScene());

          });

        this._startButton.setEffect(new DropShadow(20, Color.WHITESMOKE));
        this._startButton.setTranslateY(_midY + 250 - 64);
        this._startButton.setTranslateX(VideoOptions.
            DEFAULT_RENDER_RESOLUTION_WIDTH
                - 240);
        this.addNodes(GameSetupScene.this._startButton);

    }

    /**
     * Sets a given Wand as the chosen one for the current choosing Player.
     *
     * @param w - the wand you have chosen.
     */
    private void chooseWand (Wand w)
    {
        GameController.getInstance().getCurrentPlayerController()
        .getPlayer(this._choosingPlayer).chooseWand(w);
        this._pWVs[this._choosingPlayer].setWand(w);
        this._choosingPlayer++;

        if (this._choosingPlayer == 2)
          {
            this._choosingPlayer = 0;
          }

        this._text.setText(SELECT_WAND + this._choosingPlayer);
    }

    /**
     * Refreshs the Scene.
     *
     * @return - a refreshed GameSetupScene.
     */
    @Override
    public ChaosScene refresh ()
    {
        return this;
    }

}
```

6.45.4 GameSetupSceneMP

Listing 187: GameSetupSceneMP.java

```
package View.Scenes;
```

```
3  import Control.Controllers.GameController;
4  import Control.Controllers.SceneController;
5  import Control.Network.GameClient;
6  import Control.Network.Packets.Packet;
7  import Model.Card.Wand;
8  import Model.Card.WandFactory;
9  import static Resource.Constants.GeneralConstants.MenuConstants.*;
10 import Resource.Images.Images;
11 import Resource.VideoOptions;
12 import View.Tooltip.TooltipBox;
13 import View.UIElements.*;
14 import java.io.FileNotFoundException;
15 import java.util.logging.Level;
16 import java.util.logging.Logger;
17 import javafx.geometry.Pos;
18 import javafx.scene.effect.DropShadow;
19 import javafx.scene.paint.Color;

21 /**
22  * The GameSetupScene displays which Wand each Player has chosen and which Wands
23  * are chosable, it also provides a button to start the game.
24  */
25 public class GameSetupSceneMP extends ChaosScene
26 {
27     /**
28      * X-Coordinate of this Scene.
29      */
30     private int _midX = 0;

32     /**
33      * Y-Coordinate of this Scene.
34      */
35     private int _midY = 0;

37     /**
38      * The most left coordinate of this Scene to Adjust the Wandview.
39      */
40     private double _leftMost = 0.0;

42     /**
43      * The Button to start the Game.
44      */
45     private RoundButton _startButton = null;

47     /**
48      * The View onto the chosable Wands.
49      */
50     private WandView[] _wands = null;

52     /**
53     * VBox where the PlayerWandViews are displayed in.
54     */
55     private ChaosVBox _pWVBox = null;

57     /**
58      * 0 if host, 1 if joining
59      */
60     private int _playerNumber = -1;

62     /**
63      * PlayerWandView is stored here.
64      */
65     private PlayerWandView _pWV = null;

67     /**
68      * The Background of this Scene.
```

```
     */
    private ChaosBackground _bg = null;

    /**
     * Shows Text with an own Background.
     */
    private TextWithBackground _text = null;

    /**
     * The GameSetupScene displays which Wand each Player has chosen and
     * which Wands are chosable, it also provides a button to start the game.
     *
     * @param playerNumber - number of the Player current choosing player.
     */
    public GameSetupSceneMP (int playerNumber)
    {

        this._startButton = new RoundButton(READY);
        this._wands = new WandView[WandFactory.getWandCount()];
        this._pWVBox = new ChaosVBox();
        _midX = (int) (VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH /2 - 289
           / 2);
        _midY = (int) (VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT / 2) - 250;

        this._leftMost = _midX - 289 * 1.5d * (_wands.length - 1)
           / 2 - 11 * (_wands.length - 1) / 2;
        this._playerNumber = playerNumber;
        this.init();
    }

    /**
     * Inits all Attributes of the Scene to display it.
     */
    public void init ()
    {
        this.initBackground();
        this.initMenuButton();
        this.initText();
        this.initWandViews();
        this.initTooltips();
        this.initPlayerWandViews();
        this.initStartButton();
    }

    /**
     * inits the Tooltips.
     */
    private void initTooltips ()
    {
        this.addNodes(TooltipBox.getBox());
    }

    /**
     * Inits the Texts.
     */
    private void initText ()
    {
        this._text = new TextWithBackground(SELECT_WAND_MULTI, 75);
        this._text.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT
           / 2 - 450);
        this._text.setTranslateX(VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH
           / 2 - this._text.getTotalWidth() * 1.5d / 2);

        this.addNodes(this._text);
    }
```

```
    /**
     * Inits the Background of this Scene.
     */
    private void initBackground ()
    {
        try
          {
            _bg = new ChaosBackground(Images.MENU3);
            this.addNodes(this._bg);
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameSetupScene.class.getName()).
                log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Inits the view on to the Wands you can choose.
     */
    private void initWandViews ()
    {
        for (int i = 0; i < this._wands.length; i++)
          {
            this._wands[i] = new WandView(i);

            this._wands[i].setScaleX(1.5d);
            this._wands[i].setScaleY(1.5d);

            this._wands[i].setTranslateY(_midY);
            this._wands[i].setTranslateX(this._leftMost + i * 289 * 1.5d
               + 16 * (i - 5));

            this._wands[i].setOnMouseClicked(event ->
              {
                this.chooseWand(((WandView) event.getSource()).getWand());
              });

            this.addNodes(this._wands[i]);
          }

    }

    /**
     * Inits the view on to the Wands the Players have chosen.
     */
    private void initPlayerWandViews ()
    {
        this._pWVBox.setAlignment(Pos.CENTER_LEFT);
        this._pWVBox.setTranslateX(80);
        this._pWVBox.setScaleX(1.5d);
        this._pWVBox.setScaleY(1.5d);
        this._pWVBox.setMouseTransparent(true);

        this._pWV = new PlayerWandView(this._playerNumber);

        this._pWVBox.getChildren().add(this._pWV.get());

        this.addNodes(this._pWVBox);
    }

    /**
     * Inits the Button to the menu.
     */
    private void initMenuButton ()
    {
```

```
        ChaosMenuButton mb = new ChaosMenuButton(BACK_TO_MAIN_MENU);
        mb.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT - 300);
        mb.setTranslateX(150);
        mb.setScaleX(2);
        mb.setScaleY(2);

        mb.setOnMouseReleased(event ->
          {
            GameController.getInstance().shutDown();
            SceneController.getInstance().setScene(new MenuScene());
          });

        this.addNodes(mb);
    }

    /**
     * Inits the Button you have to press to start the Game.
     */
    private void initStartButton ()
    {

        this._startButton.setOnMouseReleased(event ->
          {
            /*
             * das macht der server!
             * MapController.getInstance().init();
             * FluxMeter.getInstance().init();
             * SoundController.getInstance().getMusic().stopMenuMusic();
             * SoundController.getInstance().getMusic().playStartNewTurn();
             */
            if (!_pWV.hasWand())
              {
                return;
              }
            else
              {
                GameClient client = GameController.getInstance().getClient();
                client.setReady(true);
                client.sendPackage(new Packet(LOBBY_READY));
              }

          });

        this._startButton.setEffect(new DropShadow(20, Color.WHITESMOKE));
        this._startButton.setTranslateY(_midY + 250 - 64);
        this._startButton.setTranslateX(VideoOptions.
            DEFAULT_RENDER_RESOLUTION_WIDTH - 240);
        this.addNodes(GameSetupSceneMP.this._startButton);
    }

    /**
     *  Sets a given Wand as the chosen one for the current choosing Player.
     *
     * @param w - the wand you have chosen.
     */
    private void chooseWand (Wand w)
    {
        this._pWV.setWand(w);
        GameController.getInstance().getCurrentPlayerController()
            .getPlayer(this._playerNumber).chooseWand(w);
    }

    /**
     * Refreshs the Scene.
     *
     * @return - a refreshed GameSetupScene.
```

```
     */
    @Override
    public ChaosScene refresh ()
    {
        return this;
    }

}
```

6.45.5 MenuScene

Listing 188: MenuScene.java

```
package View.Scenes;

import View.Menus.MainMenu;
import View.Tooltip.TooltipBox;

/**
 * The MenuScene appears if a player presses esc while the game runs. There
 * different options, you can resume to the game, go back to the main menu and
 * end the game or adjust the sound.
 */
public class MenuScene extends ChaosScene
{
    /**
     * The MainMenu that will be displayed in this scene.
     */
    private MainMenu _mainMenu = null;

    /**
     * The MenuScene appears if a player presses esc while the game runs. There
     * are different options, you can resume to the game, go back to the main
     * menu and end the game or adjust the sound.
     */
    public MenuScene ()
    {
        this._mainMenu = new MainMenu(this);
        this.addNodes(_mainMenu);
        this.init();
    }

    /**
     * Inits the Box of this Scenes Tooltip.
     */
    public void init ()
    {
        TooltipBox box = new TooltipBox();
    }

    /**
     * refreshs the Scene.
     *
     * @return - a refreshed Scene.
     */
    @Override
    public ChaosScene refresh ()
    {
        return this;
    }
}
```

6.46 View.Stylesheets

6.46.1 MenuButton

Listing 189: MenuButton.css

```
.menu-button
{
    -fx-background-color: lightgray;
    -fx-text-fill: white;
}
```

6.46.2 MenuItem

Listing 190: MenuItem.css

```
.menu-item
{
    -fx-background-color: gray;
    -fx-text-fill: white;
}
```

6.46.3 MenuSlider

Listing 191: MenuSlider.css

```
.slider
{
    -fx-show-tick-marks: true;
    -fx-major-tick-unit: 5;
    -fx-snap-to-ticks: true;
}

.slider .thumb
{
    -fx-padding: 15;
    -fx-background-color: white;
}

.slider .track
{
    -fx-padding: 15;
    -fx-background-color: rgba(0,0,0,.6);
}
```

6.46.4 MenuTextField

Listing 192: MenuTextField.css

```
.text-field
{
    -fx-border-color: darkgray;
    -fx-background-color: gray;
    -fx-text-fill: white;
}
```

6.47 View.Tooltip

6.47.1 CardTooltip

Listing 193: CardTooltip.java

```
package View.Tooltip;

import Model.Card.ICard;
import javafx.scene.effect.DropShadow;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;

/**
 * Card Tooltips provide information about the cards in the players hand.
 */
public class CardTooltip extends Tooltip
{

    /**
     *
     */
    private int _descriptionLength = 1;

    /**
     *
     */
    private ICard _card = null;

    /**
     * Card Tooltips provide information about the cards in the players hand.
     * this constructor sets the title of this Tooltip to the given title.
     *
     * @param title -
     * @param c     -
     */
    public CardTooltip (String title, ICard c)
    {
        super(title);
        this._card = c;
        this.initialize();
    }

    /**
     * Adds all relevant Information to this Tooltip.
     */
    private void initialize ()
    {
        this.addDefaultInfo();
        this.addDescription();
    }

    /**
     * Adds Information about ManaCost and CardType to the Tooltip.
     */
    private void addDefaultInfo ()
    {
        this.addInfo("Cost", ((Integer) this._card.getManaCost()).toString());
        this.addInfo("Type", this._card.getType().toString());
    }

    /**
     * Adds this Tooltips cards Description to the Tooltip.
     */
    private void addDescription ()
    {
        Text description = new Text();
```

```
        description.setText(formattedDescription(this._card.getDescription()));
        description.setFill(Color.WHITE);
        description.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 12));
        description.setTextAlignment(TextAlignment.LEFT);
        description.setEffect(new DropShadow(5, Color.GHOSTWHITE));
        description.setTranslateY(super.getLength() + 20);
        description.setTranslateX(10);

        this.addNode(description);
    }

    /**
     * Formats the given String to fit the width of the Tooltip.
     *
     * @param d - The unformatted Description.
     *
     * @return  - the formatted Description.
     */
    private String formattedDescription (String d)
    {
        String result = "";
        while (d.length() > 0)
          {
            if (d.length() > 50)
              {
                this._descriptionLength++;

                //Cuts The String after 49 letters.
                result += d.substring(0, 49) + "\n";
                d = d.substring(49);
              }
            else
              {
                result += d.substring(0, d.length());
                d = "";
              }
          }
        return result;
    }

    /**
     * Calculates the Length of this Tooltip in pixels. Adds the length being
     * added by the Description to the already existing length of the Tooltip.
     *
     * @return - the Length of this Tooltip in pixels.
     */
    @Override
    protected int getLength ()
    {
        return super.getLength() + 20 + this._descriptionLength * 20;
    }

}
```

6.47.2 GOTooltip

Listing 194: GOTooltip.java

```
package View.Tooltip;

/**
 * GOTooltips provide Information about default GameObjects such as Obstacles.
 */
public class GOTooltip extends Tooltip
{
```

```
    /**
     * GOTooltips provide Information about default GameObjects such as
     * Obstacles.
     * this constructor sets this Tooltips title to the given title.
     *
     * @param title - The given title.
     */
    public GOTooltip (String title)
    {
        super(title);
        this.init();
    }
}
```

6.47.3 MinionTooltip

Listing 195: MinionTooltip.java

```
package View.Tooltip;

import Model.GameObject.Attribute;
import Model.GameObject.CreatureAdapter;
import javafx.scene.effect.DropShadow;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;

/**
 * MinionTooltips exist to describe a Creature when the Tile containing the
 * Creature is selected.
 */
public class MinionTooltip extends Tooltip
{
    /**
     *
     */
    private CreatureAdapter _minion = null;

    /**
     * MinionTooltips exist to describe a Creature when the Tile containing the
     * Creature is selected.
     * this constructor sets this Tooltips title to the given String.
     *
     * @param title -  The Title will be set to this String instead of the
     *                 Creatures name.
     * @param minion - The Tooltip will describe this Creature.
     */
    public MinionTooltip (String title, CreatureAdapter minion)
    {
        super(title);
        this._minion = minion;
        addAllMinionValues(minion);
        this.init();
    }

    /**
     * Adds all Information about the Creature to this Tooltip.
     *
     * @param minion - The Information will be accessed from this Creature.
     */
    private void addAllMinionValues (CreatureAdapter minion)
    {
        this.addDefaultInformation(minion);
```

```
        //Checks if the Creature has any Attributes
        if (!minion.getAttributes().isEmpty())
          {
            this.addAttributeTitle();

            //Cycles through all Attributes and adds all of them to the Tooltip.
            for (int i = 0; i < minion.getAttributes().size(); i++)
              {
                this.addAttributeText(minion.getAttributes().get(i), i);
              }
          }
    }

    /**
     * Adds all by default relevant Information about the Creature to this
     * Tooltip.
     * Including:
     * -Owner Number
     * -Health
     * -Melee Attack Damage
     * -Attack Range
     * -Physical Resistance
     * -Magical Resistance
     * -Remaining Movementpoints
     *
     * @param minion - The Information will be accessed from this Creature.
     */
    private void addDefaultInformation (CreatureAdapter minion)
    {
        this.addInfo("Owner", ((Integer) (minion.getOwner() + 1)).toString());
        this.addInfo("Health", minion.getHealth() + " / " +
            minion.getMaxHealth());
        this.addInfo("Melee", minion.getAttack() + " Ranged : " +
            minion.getRangedAttack());
        this.addInfo("Attack Range", ((Integer) minion.getAttackRange())
            .toString());
        this.addInfo("Physical Resistance", ((Integer) minion._getPhysRes())
            .toString());
        this.addInfo("Magical Resistance", ((Integer) minion._getMagicRes())
            .toString());
        this.addInfo("Movementpoints", minion.getMovementpoints() + " / " +
            minion.getMaxMvp());
    }

    /**
     * Adds a underlined Title above all Attributes to the Tooltip.
     */
    private void addAttributeTitle ()
    {
        Text aText = new Text("Attributes:");

        aText.setFill(Color.WHITE);
        aText.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 15));
        aText.setUnderline(true);
        aText.setTextAlignment(TextAlignment.LEFT);
        aText.setEffect(new DropShadow(3, Color.GHOSTWHITE));

        aText.setTranslateY(super.getLength() + 20);
        aText.setTranslateX(10);

        this.addNode(aText);
    }

    /**
     * Adds Text describing one Attribute to the Tooltip.
     *
```

```
     * @param att      - The Attribute being described
     * @param position - The Position in the List of Attributes
     */
    private void addAttributeText (Attribute att, int position)
    {
        Text atText = new Text(att.toString());

        atText.setFill(Color.WHITE);
        atText.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 12));
        atText.setTextAlignment(TextAlignment.LEFT);
        atText.setEffect(new DropShadow(2, Color.GHOSTWHITE));

        atText.setTranslateY(super.getLength() + 40 + 20 * position);
        atText.setTranslateX(10);

        this.addNode(atText);
    }

    /**
     * Refreshes The Minion Values. To be used after the Values have changed in
     * any way.
     */
    @Override
    protected void refresh ()
    {
        this.clearInfo();
        addAllMinionValues(this._minion);
    }

    /**
     * Calculates the Length of this Tooltip in pixels. Adds the length being
     * added by the Attributes to the already existing length of the Tooltip.
     *
     * @return - the Length of this Tooltip in pixels.
     */
    @Override
    protected int getLength ()
    {
       return super.getLength() + this._minion.getAttributes().size() * 20 + 40;
    }

}
```

6.47.4 TileTooltip

Listing 196: TileTooltip.java

```
package View.Tooltip;

import Model.Map.Tile;

/**
 * Tile Tooltips provide Information about the selected Tile such as its height.
 */
public class TileTooltip extends Tooltip
{

    /**
     * Tile Tooltips provide Information about the selected Tile such as its
     * height.
     * this constructor adds information about the given tile
     *
     * @param tile - The given tile.
     */
    public TileTooltip (Tile tile)
    {
```

```
        super("Tile");
        this.addInfo("Height", ((Integer) tile.getHeight()).toString());
    }

    /**
     * Removes the Tooltip about the previously clicked tile before showing.
     */
    @Override
    public void show ()
    {
        this.clear();
        super.show();
    }

}
```

6.47.5 Tooltip

Listing 197: Tooltip.java

```
package View.Tooltip;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import javafx.application.Platform;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.GaussianBlur;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;

/**
 * Tooltips provide information about the object that is currently in the
 * players focus.
 */
public abstract class Tooltip extends StackPane
{

    /**
     * Default title.
     */
    protected String _title = null;

    /**
     * The Values of a Tooltip are the information that is to be displayed.
     * The Key defines the type of information, and the value is the
     * corresponding information.
     */
    protected HashMap<String, String> _values = new HashMap<String, String>();

    /**
     * A list of nodes separate from the .getChildren() inherited from
     * StackPane.
     */
    protected List<Node> _children = new ArrayList<Node>();

    /**
     * Tooltips provide information about the object that is currently in the
     * players focus.
```

```
     */
    public Tooltip ()
    {

        this._title = "Undefined Tooltip";

    }
    /**
     * Tooltips provide information about the object that is currently in the
     * players focus.
     *
     * @param title - The title defines the Object that the Tooltip is describing.
     */
    public Tooltip (String title)
    {

        this._title = title;

    }

    /**
     * Tooltips provide information about the object that is currently in the
     * players focus.
     *
     * @param title  - The title defines the Object that the Tooltip is
     *                 describing.
     * @param values - The Values describe the Object.
     */
    public Tooltip (String title, HashMap<String, String> values)
    {

        this._title = title;
        this._values = values;

    }

    /**
     * Setter.
     *
     * @param title - This String will be the new Title of this Tooltip.
     */
    public void setTitle (String title)
    {
        this._title = title;
        this.init();

    }

    /**
     * Calculates the length of the Tooltip in pixels. Relevant for setting the
     * size of the Background rectangle.
     *
     * @return - the length of the Tooltip in pixels.
     */
    protected int getLength ()
    {
        return 50 + this._values.size() * 20;
    }

    /**
     * Initializes all elements of this Tooltip and adds them to the StackPane.
     */
    protected void init ()
    {
        this.getChildren().clear();
```

```
        this.setMouseTransparent(true);
        this.setAlignment(Pos.TOP_LEFT);

        this.initBackground();
        this.initTitle();
        this.initProperties();

        this.getChildren().addAll(_children);

    }

    /**
     * Initializes the Background Rectangle.
     */
    private void initBackground ()
    {
        Rectangle bg = new Rectangle(350, this.getLength());

        bg.setOpacity(0.6);
        bg.setFill(Color.BLACK);
        bg.setEffect(new GaussianBlur(3.5));
        bg.setEffect(new DropShadow(20, Color.GHOSTWHITE));

        this.getChildren().add(bg);
    }

    /**
     * Initializes the Title.
     */
    private void initTitle ()
    {
        Text title = new Text(this._title);

        title.setFill(Color.WHITE);
        title.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 24));
        title.setUnderline(true);
        title.setTextAlignment(TextAlignment.LEFT);
        title.setEffect(new DropShadow(10, Color.GHOSTWHITE));

        title.setTranslateY(10);
        title.setTranslateX(10);

        this.getChildren().add(title);
    }

    /**
     * Initializes all given Information Text on this Tooltip.
     */
    private void initProperties ()
    {
        if (!this._values.isEmpty())
          {
            int c = 0;
            for (String s : this._values.keySet())
              {
                c++;

                //This will look like the following example: "Health : 30"
                Text t = new Text(s + " : " + this._values.get(s));

                t.setFill(Color.WHITE);
                t.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 12));
                t.setTextAlignment(TextAlignment.LEFT);
                t.setEffect(new DropShadow(5, Color.GHOSTWHITE));

                /**
```

```
                    * All Information is placed manually 20 pixels below the one
                    * above. This is done manually to ensure that we calculate the
                    * length of the Tooltip correctly.
                    */
                   t.setTranslateY(30 + 20 * c);

                   t.setTranslateX(10);

                   this.getChildren().add(t);
                }
            }
        }

        /**
         * To be overwritten
         */
        protected void refresh ()
        {

        }

        /**
         * Removes all information from this Tooltip.
         */
        protected void clearInfo ()
        {
            this._children.clear();
            this._values.clear();
        }

        /**
         * Removes all Tooltips from the main TooltipBox.
         */
        protected void clear ()
        {
            TooltipBox.getBox().getChildren().clear();
        }

        /**
         * Adds a node to this Tooltip.
         *
         * @param n - the node to be added.
         */
        public void addNode (Node n)
        {
            this._children.add(n);
            this.init();

        }

        /**
         * Adds a piece of Information to the HashMap of Values.
         * These will be displayed as in the following example:
         * "Health : 30"
         *
         * @param key   - The Key Defines the information e.g. ("Health")
         * @param value - The Value holds the information e.g. ("30")
         */
        public void addInfo (String key, String value)
        {
            this._values.put(key, value);
            this.init();
        }

        /**
         * Adds this Tooltip to the main TooltipBox.
```

```
245      */
246     public void show ()
247     {
248         this.hide();
249         this.refresh();
250         TooltipBox.getBox().add(this);
251     }

253     /**
254      * Adds this Tooltip to the main TooltipBox but also ensures that this is
255      * executed on the main javaFX Thread.
256      *
257      * @param tip - this Tooltip will be added to the TooltipBox.
258      */
259     public void show (Tooltip tip)
260     {
261         Platform.runLater(() ->
262           {
263             tip.hide();
264             tip.refresh();
265             TooltipBox.getBox().add(tip);
266           });
267     }

269     /**
270      * Removes this Tooltip from the main TooltipBox.
271      */
272     public void hide ()
273     {
274         TooltipBox.getBox().remove(this);
275     }

277     /**
278      * Removes this Tooltip from the main TooltipBox but also ensures that this
279      * is executed on the main javaFX Thread.
280      *
281      * @param tip - this Tooltip will be removed from the TooltipBox.
282      */
283     public void hide (Tooltip tip)
284     {
285         Platform.runLater(() ->
286           {
287             TooltipBox.getBox().remove(tip);
288           });
289     }

291 }
```

6.47.6 TooltipBox

Listing 198: TooltipBox.java

```
1  package View.Tooltip;

3  import Resource.Dimensions;
4  import Resource.VideoOptions;
5  import View.UIElements.IChaosUI;
6  import javafx.scene.layout.VBox;

8  /**
9   * the TooltipBox holds all Tooltips and organizes them Vertically under each
10  * other.
11  */
12 public class TooltipBox extends VBox implements IChaosUI
13 {
14     /**
```

```
     *
     */
    private boolean hasBeenScaled = false;

    /**
     *
     */
    private static TooltipBox _instance;

    /**
     * Singleton getter.
     *
     * @return - the TooltipBox instance.
     */
    public static TooltipBox getBox ()
    {
        return _instance;
    }

    /**
     * This Constructor is to be called once at the start of the game. It
     * initializes the TooltipBox that holds all Tooltips into the top right
     * corner of the screen.
     */
    public TooltipBox ()
    {

        this.setMinSize(350, 500);
        this.setSpacing(30);
        this.setScaleX(1.75d);
        this.setScaleY(1.75d);

        /*
         *   Sets the Origin of the tooltip in a way,
         *   that it is in the top right corner of the screen.
         */
        double originX = (VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH - 350
            * 1.75d);
        double originY = 100;

        this.setTranslateX(originX);
        this.setTranslateY(originY);

        _instance = this;
    }

    /**
     * Adds a Tooltip to the Box.
     *
     * @param tip - This Tooltip will be added to the Box.
     */
    public void add (Tooltip tip)
    {
        this.getChildren().add(tip);
    }

    /**
     * removes a Tooltip from the Box.
     *
     * @param tip - This Tooltip will be removed from the Box.
     */
    public void remove (Tooltip tip)
    {
        if (this.getChildren().contains(tip))
          {
            this.getChildren().remove(tip);
```

```
            }
    }

    /**
     * Scales this TooltipBox to the Size of the monitor in use.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        if (!this.hasBeenScaled)
          {
            //gets the current monitors height and width.
            double monitorHeight = Dimensions.getHeight();
            double monitorWidth = Dimensions.getWidth();

            //The Original size of the renderes screen.
            double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
            double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

            /*
             * calculates the difference in height
             * and width between both resolutions.
             */
            double heightDifference = originalHeight - monitorHeight;
            double widthDifference = originalWidth - monitorWidth;

            //Scales this Node appropriately.
            this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
            this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

            //Positions this Node at the appropriate position.
            this.setTranslateY(this.getTranslateY() - heightDifference
                    * (this.getTranslateY() / originalHeight));
            this.setTranslateX(this.getTranslateX() - widthDifference
                    * (this.getTranslateX() / originalWidth));

            //Makes Sure that this node can not be scaled twice.
            this.hasBeenScaled = true;
          }
    }
}
```

6.47.7 WandTooltip

Listing 199: WandTooltip.java

```
package View.Tooltip;

/**
 * Wand Tooltips provide information about a Wand, in the Game Setup Scene, such
 * as how many Cards of which type are available with each wand.
 */
public class WandTooltip extends Tooltip
{

    /**
     * Wand Tooltips provide information about a Wand, in the Game Setup Scene,
     * such
     * as how many Cards of which type are available with each wand.
     * this constructor sets this Tooltip's title to the given title.
     *
     * @param s - The given title.
     */
    public WandTooltip (String s)
    {
        super(s);
```

```
    }
}
```

6.48 View.UIElements

6.48.1 CardView

Listing 200: CardView.java

```
package View.UIElements;

import Control.Controllers.SceneController;
import Control.Controllers.TurnController;
import Control.SemiControllers.Selector;
import Model.Card.ICard;
import Resource.Images.Images;
import View.Scenes.GameScene;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.effect.ColorAdjust;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;

/**
 * The CardView displays one Card in the Hand of the active player, as well as
 * its manacost.
 */
public class CardView extends StackPane
{
    /**
     *
     */
    private ICard _card = null;

    /**
     *
     */
    private ImageView _cardImage = null;

    /**
     *
     */
    private ImageView _manaBG = null;

    /**
     *
     */
    private Text _manaText = null;

    /**
     *
     */
    private ColorAdjust _effect = null;

    /**
     * This is only relevant for multiplayer, as you should not be able to see
```

```
     * your opponents cards.
     * - True: this CardView belongs to a Card in the opponents hand and should
     * not be revealed.
     * - False: this CardView belongs to the Player himself and should be
     * displayed as normal.
     */
    private boolean _opponent = false;

    /**
     * The CardView displays one Card in the Hand of the active player, as well
     * as its manacost.
     *
     * @param im       - This Image should correspond to the referring card
     * @param c        - This CardView will refer to this card
     * @param opponent - False - this CardView should not reveal the Card.
     *                   True - this CardView should reveal the Card.
     */
    public CardView (Image im, ICard c, boolean opponent)
    {
        this.init(im, c, opponent);
    }

    /**
     * Sets all relevant default values for this CardView
     *
     * @param im       - The Corresponding image.
     * @param c        - The Corresponding card.
     * @param opponent - False - this CardView should not reveal the Card.
     *                   True - this CardView should reveal the Card.
     */
    private void init (Image im, ICard c, boolean opponent)
    {
        this._effect = new ColorAdjust();
        this._opponent = opponent;
        this.initImage(im);

        if (!this._opponent)
          {
            //Everything in here is only relevant if the CardView corresponds
            //to a card belonging to the local player.
            this.initManaBG();
            this.initManaText();

            this._card = c;

            this.setOnMouseClicked(event ->
              {
                System.out.println(this._card.toString());

                if (event.getButton() == MouseButton.PRIMARY)
                  {
                    Selector.getInstance().selectCard(c);
                  }

                SceneController.getInstance().refresh();
              });
          }

    }

    /**
     * Sets and scales the corresponding Image.
     *
     * @param im - the corresponding Image.
     */
    private void initImage (Image im)
```

```
    {
        this._cardImage = new ImageView(im);

        this._cardImage.setScaleX(2.3d);
        this._cardImage.setScaleY(2.3d);
        this._cardImage.setMouseTransparent(true);
        this.getChildren().add(this._cardImage);
    }

    /**
     * Sets and Scales the Background for the manaText.
     */
    private void initManaBG ()
    {
        try
          {
            this._manaBG = new ImageView(new Image(new FileInputStream(Images
                .MANA_COST)));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameScene.class.getName())
                .log(Level.SEVERE, null, ex);
          }

        this._manaBG.setTranslateX(35);
        this._manaBG.setTranslateY(42);
        this._manaBG.setMouseTransparent(true);
        this._manaBG.setScaleX(1.3f);
        this._manaBG.setScaleY(1.3f);
        this._manaBG.toFront();
        this._manaBG.setEffect(_effect);
        this.getChildren().add(this._manaBG);
    }

    /**
     * Sets and Scales the text corresponding to the manaCost of this CardView's
     * card.
     */
    private void initManaText ()
    {
        this._manaText = new Text();
        this._manaText.setTranslateX(35);
        this._manaText.setTranslateY(42);
        this._manaText.setMouseTransparent(true);
        this._manaText.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 24));
        this._manaText.toFront();
        this.getChildren().add(this._manaText);
    }

    /**
     * refreshes the Color of this CardViews manaText if the player does not
     * have enough mana to cast the corresponding card.
     */
    private void refreshHue ()
    {
        if (this._card.getManaCost() > TurnController.getInstance().
                getCurrentPlayer().getMana())
          {
            //Shifts the Hue in a way, that makes it red(-ish).
            this._effect.setHue(-0.1);
            this._effect.setContrast(0.8);
            this._effect.setSaturation(0.8);
            this._manaText.setFill(Color.BLACK);
            _manaText.setEffect(new DropShadow(3, Color.RED));
          }
```

```
        else
          {
            this._effect.setHue(0);
            this._effect.setContrast(0);
            this._effect.setSaturation(0);
          }
    }

    /**
     * refreshes the text showing the manaCost of the corresponding card,
     * showing it as green, if the manacost is decreased due to flux shift and
     * red if it has increased.
     */
    private void refreshManaText ()
    {

        this._manaText.setText(((Integer) this._card.getManaCost()).toString());

        switch (this._card.getManaCostIncreased())
          {
            case 1:
                _manaText.setFill(Color.RED);
                _manaText.setEffect(new DropShadow(10, Color.LIGHTCORAL));
                break;
            case -1:
                _manaText.setFill(Color.GREEN);
                _manaText.setEffect(new DropShadow(10, Color.LIGHTGREEN));
                break;
            case 0:
                _manaText.setFill(Color.BLUE);
                _manaText.setEffect(new DropShadow(10, Color.AQUA));
                break;
          }

    }

    /**
     * Refreshes the text showing the manaCost of the corresponding card, as
     * well as its ColorShift in some cases.
     */
    public void refresh ()
    {
        if (!this._opponent)
          {
            this.refreshManaText();
            this.refreshHue();
          }
    }
}
```

6.48.2 ChaosBackground

Listing 201: ChaosBackground.java

```
package View.UIElements;

import Resource.Dimensions;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;

/**
 * A ChaosBackground is to be the Image in the Background of any Scene, we
 * implement IChaosUI, to be able to scale this Image to the current
```

```
 * Monitors size.
 */
public class ChaosBackground extends StackPane implements IChaosUI
{
    /**
     *
     */
    private ImageView _image = null;

    /**
     * A ChaosBackground is to be the Image in the Background of any Scene, we
     * implement IChaosUI, to be able to scale this Image to the current
     * Monitors size.
     *
     * @param image - The File path to the Background Image.
     *
     * @throws FileNotFoundException if there is no File at the target location
     *                               of the image string.
     */
    public ChaosBackground (String image) throws FileNotFoundException
    {
        this._image = new ImageView(new Image(new FileInputStream(image)));

        this.getChildren().add(this._image);
    }

    /**
     * Inherited from IChaosUI.
     *
     * called once at the start of a scene.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        //sets the Images size to the current monitors height and width.
        this._image.setFitHeight(Dimensions.getHeight());
        this._image.setFitWidth(Dimensions.getWidth());
    }
}
```

6.48.3 ChaosMenuButton

Listing 202: ChaosMenuButton.java

```
package View.UIElements;

import Resource.Dimensions;
import Resource.VideoOptions;
import javafx.geometry.Pos;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.GaussianBlur;
import javafx.scene.effect.Glow;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;

/**
 * The ChaosMenuButton is our Implementation of a Button that is used in our
 * Menus.
 */
public class ChaosMenuButton extends StackPane implements IChaosUI
{

    /**
     *
```

```
     */
    private Text _text = null;

    /**
     *
     */
    private Rectangle _bg = null;

    /**
     * The ChaosMenuButton is our Implementation of a Button that is used in our
     * Menus.
     *
     * @param name - Text displayed on button.
     */
    public ChaosMenuButton (String name)
    {
        this.init(name);
    }

    /**
     * sets all default values.
     *
     * @param name - Text displayed on button.
     */
    private void init (String name)
    {

        this.initBG();
        this.initText(name);
        this.setMouseEvents();

        //Menu buttons in my menu are left
        this.setAlignment(Pos.CENTER_LEFT);
        //button slightly rotated to make the menu look fancy
        this.setRotate(-2);
    }

    /**
     * sets _text & font
     *
     * @param name The text on the button.
     */
    private void initText (String name)
    {
        this._text = new Text(name);
        this._text.setFont(this._text.getFont().font(20));
        this._text.setFill(Color.WHITE);
        this.getChildren().add(this._text);
    }

    /**
     * sets the background rectangle.
     */
    private void initBG ()
    {
        this._bg = new Rectangle(250, 30);
        this._bg.setOpacity(0.6);
        this._bg.setFill(Color.BLACK);
        //add blur for fancy looks
        this._bg.setEffect(new GaussianBlur(3.5));
        this.getChildren().add(this._bg);
    }

    /**
     * Sets all MouseEvents relevant for highlighting this button.
     */
```

```
    private void setMouseEvents ()
    {
        /*
         * Mouse entered event: moves button to the right and swaps background
         * and
         * _text color to indicacte which button the mouse is on (for fancy
         * looks)
         */
        this.setOnMouseEntered(event ->
          {
            this._bg.setTranslateX(10);
            this._text.setTranslateX(10);
            this._bg.setFill(Color.WHITE);
            this._text.setFill(Color.BLACK);
          });

        /*
         * Mouse exited event: moves button back to the left and swaps
         * background and
         * _text color to indicacte which button the mouse is on (for fancy
         * looks). Basically undoes changes from mouseEntered.
         */
        this.setOnMouseExited(event ->
          {
            this._bg.setTranslateX(0);
            this._text.setTranslateX(0);
            this._bg.setFill(Color.BLACK);
            this._text.setFill(Color.WHITE);
          });

        //On click visual effect...
        DropShadow drop = new DropShadow(50, Color.WHITE);
        drop.setInput(new Glow());
        //...which is  used in  the onMousePressed event...
        this.setOnMousePressed(event -> setEffect(drop));
        //...and then undone on mouse Release
        this.setOnMouseReleased(event -> setEffect(null));
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        //gets the current monitors height and width.
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        //The Original size of the renderes screen.
        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        //calculates the difference in height and width between both resolutions.
        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        //Scales this Node appropriately.
        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        //Positions this Node at the appropriate position.
        this.setTranslateY(this.getTranslateY() - heightDifference *
                 (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference *
```

```
                    (this.getTranslateX() / originalWidth));
    }

}
```

6.48.4 ChaosVBox

Listing 203: ChaosVBox.java

```
package View.UIElements;

import Resource.Dimensions;
import Resource.VideoOptions;
import javafx.scene.layout.VBox;

/**
 * VBox implementation that can be scaled to monitor size.
 */
public class ChaosVBox extends VBox implements IChaosUI
{

    /**
     * Inherited from IChaosUI
     * Scales this VBox to the Size of the monitor in use.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        //gets the current monitors height and width.
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        //The Original size of the renderes screen.
        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        //calculates the difference in height and width between both resolutions.
        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        //Scales this Node appropriately.
        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        //Positions this Node at the appropriate position.
        this.setTranslateY(this.getTranslateY() - heightDifference
                * (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference
                * (this.getTranslateX() / originalWidth));
    }

}
```

6.48.5 DeckUI

Listing 204: DeckUI.java

```
package View.UIElements;

import Control.Controllers.TurnController;
import Resource.Dimensions;
import Resource.Images.Images;
import Resource.VideoOptions;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;

/**
 * The DeckUI displays The amount of Cards left in the deck of the active
 * player.
 */
public class DeckUI extends StackPane implements IChaosUI
{
    /**
     *
     */
    private ImageView _iv = null;

    /**
     *
     */
    private Text _text = null;

    /**
     * The DeckUI is used solely to show how many cards a Player has left in his
     * Deck.
     */
    public DeckUI ()
    {
        this._text = new Text();
        //Sets all default values for this Node.
        this.init();
    }

    /**
     * sets all default values.
     */
    private void init ()
    {
        //Positions this Node left of the HandView.
        this.setTranslateX(VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH / 2 -
            1050);
        this.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT - 450);

        //Scales this Node to an appropriate size.
        this.setScaleX(1.5d);
        this.setScaleY(1.5d);

        try
          {
            this._iv = new ImageView(new Image(new FileInputStream(Images.DECK)));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(DeckUI.class.getName()).log(Level.SEVERE, null, ex);
          }

        this._text.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 60));
        this._text.setEffect(new DropShadow(5, Color.GREY));
        this._text.setFill(Color.BLACK);

        //Positions the Text at the appropriate position on the Background Image.
```

```
        this._text.setTranslateX(35);
        this._text.setTranslateY(40);

        this._iv.setTranslateY(50);

        this.setEffect(new DropShadow(10, Color.WHITESMOKE));

        this.getChildren().add(this._iv);
        this.getChildren().add(this._text);
    }

    /**
     * refreshes the Text representing the amount of Cards left in the Current
     * players Hand.
     */
    public void refresh ()
    {
        int deckSize = TurnController.getInstance().getCurrentPlayer().getHand()
            .getDeck().getSize();

        this._text.setText(((Integer) deckSize).toString());
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        //gets the current monitors height and width.
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        //The Original size of the renderes screen.
        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        //calculates the difference in height and width between both resolutions.
        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        //Scales this Node appropriately.
        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        //Positions this Node at the appropriate position.
        this.setTranslateY(this.getTranslateY() - heightDifference
                * (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference
                * (this.getTranslateX() / originalWidth));

        /*
         *  Repositions the Image and _text.
         *  Relevant for Monitorssmaller than 2560 x 1440 pixels.
         */
        this._iv.setTranslateY(this._iv.getTranslateY() - heightDifference
                * (this._iv.getTranslateY() / originalHeight));
        this._text.setTranslateY(this._text.getTranslateY() - heightDifference
                * (this._text.getTranslateY() / originalHeight));

    }

}
```

6.48.6 FluxBar

Listing 205: FluxBar.java

```
package View.UIElements;

import Model.Game.FluxMeter;
import Resource.Constants.GeneralConstants.GameConstants;
import static Resource.Constants.GeneralConstants.UIConstants.FLUX_BAR_Y;
import Resource.Dimensions;
import Resource.Images.Images;
import Resource.VideoOptions;
import View.Scenes.GameScene;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.geometry.Pos;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.RadialGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;

/**
 * The FluxBar gives a visual representation of the FluxShift affecting the
 * manaCost of all cards.
 */
public class FluxBar extends StackPane implements IChaosUI
{

    /**
     *
     */
    private ImageView _fluxBar = null;

    /**
     *
     */
    private Text _fText = null;

    /**
     *
     */
    private Rectangle _bar = null;

    /**
     *
     */
    private int _x = 0;

    /**
     *
     */
    private int _y = 0;

    /**
     *
```

```
     */
    private RadialGradient _gradientBlue = null;

    /**
     *
     */
    private RadialGradient _gradientRed = null;

    /**
     * The FluxBar gives a visual representation of the FluxShift affecting the
     * manaCost of all cards.
     */
    public FluxBar ()
    {
        this._fText = new Text();
        this._bar = new Rectangle();
        this._y = FLUX_BAR_Y;
        this.init();
    }

    /**
     * Sets all relevant default values.
     */
    private void init ()
    {
        try
          {
            this._fluxBar = new ImageView(new Image(new
                                        FileInputStream(Images.FLUX_BAR)));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameScene.class.getName())
                                            .log(Level.SEVERE, null, ex);
          }

        this.setScaleX(1.75d);
        this.setScaleY(1.75d);

        /**
         * Calculates the x-coordinate where this node has to be placed to
         * appear in the center of the screen.
         */
        this._x = (int) ((VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH / 2)
                                                        - (512 * 1.75d / 2));

        this.setTranslateY(this._y);
        this.setTranslateX(this._x);

        this.setAlignment(Pos.CENTER_LEFT);

        this.initText();
        this.initBar();
        this.initGradients();

        this.getChildren().add(this._fluxBar);
    }

    /**
     * Sets and positions the fluxBar text.
     */
    private void initText ()
    {
        this._fText.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 30));
        this._fText.setTextAlignment(TextAlignment.CENTER);
        this._fText.setEffect(new DropShadow(10, Color.GRAY));
```

```
        this._fText.setTranslateY(40);
        this._fText.setTranslateX(216);

        this.getChildren().add(this._fText);
    }

    /**
     * Sets and positions the actual bar.
     */
    private void initBar ()
    {
        this._bar.setHeight(16);
        this._bar.setArcHeight(10);
        this._bar.setArcWidth(10);
        this._bar.setTranslateY(0);
        this._bar.setTranslateX(256);

        this.getChildren().add(this._bar);
    }

    /**
     * initializes the color gradients for the bar.
     */
    private void initGradients ()
    {
        this.initBlueGradient();
        this.initRedGradient();
    }

    /**
     * initializes a blue color gradient.
     */
    private void initBlueGradient ()
    {

        Stop[] stops = new Stop[]
          {
            new Stop(0, Color.AQUA), new Stop(1, Color.DARKBLUE)
          };
        this._gradientBlue = new RadialGradient(0, 0, 0, 0, 1, true,
                                                CycleMethod.NO_CYCLE, stops);
    }

    /**
     * initializes a red color gradient.
     */
    private void initRedGradient ()
    {
        Stop[] stops = new Stop[]
          {
            new Stop(0, Color.RED), new Stop(1, Color.DARKRED)
          };
        this._gradientRed = new RadialGradient(0, 0, 0, 0, 1, true,
                                                CycleMethod.NO_CYCLE, stops);
    }

    /**
     * refreshes color of the bar aswell as its size.
     */
    private void refreshBarAndColors ()
    {
        switch (FluxMeter.getInstance().getLean())
          {
            case 0:
                this._bar.setFill(Color.YELLOW);
```

```
                this._fText.setFill(Color.YELLOW);
                break;
            case 1:
                this._bar.setFill(this._gradientBlue);
                this._bar.setWidth(Math.abs(FluxMeter.getInstance().getFlux() *
                    220 / GameConstants.MAX_FLUX));

                this._fText.setFill(Color.AQUA);
                break;
            case -1:
                this._bar.setFill(this._gradientRed);
                this._bar.setWidth(Math.abs(FluxMeter.getInstance().getFlux()) *
                    220 / GameConstants.MAX_FLUX);
                this._bar.setTranslateX(259 - (Math.abs(FluxMeter.getInstance()
                    .getFlux()) * 220 / GameConstants.MAX_FLUX));

                this._fText.setFill(Color.RED);
                break;
        }
    }

    /**
     * refreshes the text of this FluxBar.
     */
    public void refresh ()
    {

        this._fText.setText("Flux: " + Math.abs(FluxMeter.getInstance().getFlux(
        )));
        this.refreshBarAndColors();
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference
                * (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference
                * (this.getTranslateX() / originalWidth));
    }
}
```

6.48.7 HandView

Listing 206: HandView.java

```
package View.UIElements;

import Control.Controllers.GameController;
import Resource.Dimensions;
```

```
import Resource.VideoOptions;
import java.util.ArrayList;
import java.util.List;
import javafx.scene.layout.HBox;

/**
 * The HandView organizes all cards in the Hand of the active player as
 * CardViews next to each other horizontally.
 */
public class HandView extends HBox implements IChaosUI
{
    /**
     *
     */
    private List<CardView> _cardViews = null;

    /**
     * The HandView organizes all cards in the Hand of the active player as
     * CardViews next to each other horizontally.
     */
    public HandView ()
    {
        this._cardViews = new ArrayList<>();
        this.init();
    }

    /**
     * Sets all default values.
     */
    private void init ()
    {
        this._cardViews = GameController.getInstance().getCurrentTurnController()
            .getCurrentPlayer().getHand().getCardViews();

        this.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT - 450);
        this.setTranslateX((VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH / 2)
                - 960);

        this.setSpacing(50);

        this.setMinSize(2250, 600);
        this.setPrefSize(2250, 600);

        this.getChildren().addAll(this._cardViews);
    }

    /**
     * Removes all CardViews that no longer exist in the HandView and adds all
     * new CardViews.
     */
    public void refresh ()
    {
        this.getChildren().clear();
        this._cardViews = GameController.getInstance().getCurrentTurnController()
            .getCurrentPlayer().getHand().getCardViews();
        this.getChildren().addAll(this._cardViews);

    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
```

```
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setMinSize(1500 * monitorWidth / originalWidth, 400 * monitorHeight
            / originalHeight);
        this.setPrefSize(1500 * monitorWidth / originalWidth, 400 *
            monitorHeight / originalHeight);

        this.autosize();

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference
            * (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference
            * (this.getTranslateX() / originalWidth));
    }
}
```

6.48.8 IChaosUI

Listing 207: IChaosUI.java

```
package View.UIElements;

/**
 * All of our custom UI Nodes implement this interface to enable them all to be
 * scaled to the size of the monitor in use.
 */
public interface IChaosUI
{

    /**
     * Scales this UIElement to fit the active monitor.
     */
    public void scaleToMonitorSize ();
}
```

6.48.9 ManaBar

Listing 208: ManaBar.java

```
package View.UIElements;

import Control.Controllers.GameController;
import Resource.Dimensions;
import Resource.Images.Images;
import Resource.VideoOptions;
import View.Scenes.GameScene;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.geometry.Pos;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
```

```
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;

/**
 * The ManaBar provides Visual intel on how much mana the active player has
 * left.
 */
public class ManaBar extends StackPane implements IChaosUI
{
    /**
     *
     */
    private int _currentMana = 0;

    /**
     *
     */
    private ImageView _manaBar = null;

    /**
     *
     */
    private Text _mText = null;

    /**
     *
     */
    private Rectangle _bar = null;

    /**
     * The ManaBar provides Visual intel on how much mana the active player has
     * left.
     */
    public ManaBar ()
    {
        this.init();
    }

    /**
     * Sets all default values.
     */
    private void init ()
    {
        this._mText = new Text();
        this._bar = new Rectangle();
        this.setTranslateX(VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH - 64
            * 1.5d - 150);
        this.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT - 128
            * 1.5d - 150);
        this.setAlignment(Pos.TOP_CENTER);

        this.setScaleX(1.5d);
        this.setScaleY(1.5d);

        this.initImage();
        this.initText();
        this.initbar();

        this.refresh();
```

```
    }

    /**
     * Sets the image.
     */
    private void initImage ()
    {
        try
          {
            this._manaBar = new ImageView(new Image
                                  (new FileInputStream(Images.MANA_BAR)));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameScene.class.getName())
                .log(Level.SEVERE, null, ex);
          }
    }

    /**
     * Sets the text showing the amount of mana left.
     */
    private void initText ()
    {
        this._mText.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 22));
        this._mText.setFill(Color.WHITESMOKE);
        this._mText.setTextAlignment(TextAlignment.CENTER);
        this._mText.setEffect(new DropShadow(5, Color.GRAY));
        this._mText.setTranslateY(-16);
    }

    /**
     * sets the bar visually representing the amount of mana left.
     */
    private void initbar ()
    {
        this._bar.setWidth(28);
        this._bar.setArcHeight(10);
        this._bar.setArcWidth(10);

        Stop[] stops = new Stop[]
          {
            new Stop(0, Color.AQUA), new Stop(1, Color.DARKBLUE)
          };

        LinearGradient lg = new LinearGradient(0, 0, 1, 0, true,
            CycleMethod.NO_CYCLE, stops);

        this._bar.setFill(lg);
    }

    /**
     * re-sets the text.
     */
    private void refreshText ()
    {
        this._mText.setText("Mana: " + this._currentMana);
    }

    /**
     * re-scales the bar.
     */
    private void refreshBar ()
    {
        this._bar.setHeight(Math.min((int) this._currentMana / 1.5f, 96));
        this._bar.setTranslateY(16 + 96 - Math.min((int) this._currentMana
```

```
                                                                / 1.5f, 96));
    }

    /**
     * Refreshes the size of the bar aswell as the text representing the amount
     * of mana the active player has left.
     */
    public void refresh ()
    {
        this.getChildren().clear();

        this._currentMana = GameController.getInstance()
            .getCurrentTurnController().getCurrentPlayer().getMana();

        this.refreshBar();
        this.refreshText();

        this.getChildren().addAll(this._bar, this._manaBar, this._mText);
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference *
            (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference *
            (this.getTranslateX() / originalWidth));
    }
}
```

6.48.10 MapView

Listing 209: MapView.java

```
package View.UIElements;

import Control.Controllers.MapController;
import Model.GameObject.CreatureAdapter;
import Model.GameObject.Obstacle;
import Model.Map.Tile;
import Resource.Dimensions;
import Resource.Images.Images;
import Resource.VideoOptions;
import View.Scenes.GameScene;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.scene.Group;
```

```
import javafx.scene.image.Image;

/**
 * The Map View is the visual representation of the Map and all of the Objects
 * that are on it.
 */
public class MapView extends Group implements IChaosUI
{

    /**
     * the following 4 values are used for dragging the map on the screen
     */
    private double _mouseX = 0.0;
    private double _mouseY = 0.0;
    private double _offX = 0.0;
    private double _offY = 0.0;

    /**
     * height and width of the map view in pixels, before any scaling is done
     */
    private int[] _pxDimensions = new int[2];

    /**
     * Group for images of TileImages. Is only drawn once.
     */
    private Group _map = new Group();

    /**
     * Group for Images of other Stuff that doesn't change over the course of a
     * game. Is only drawn once. Only used for Obstacles right now.
     */
    private Group _objects = new Group();

    /**
     * Group for colored frames around each players' creatures. Redrawn on
     * refresh.
     */
    private Group _frames = new Group();

    /**
     * Group for Images of all different types of highlights. Redrawn on
     * refresh.
     */
    private Group _highlights = new Group();

    private HashMap<Tile, TileView> _tileMap = new HashMap<>();

    /**
     * The Map View is the visual representation of the Map and all of the
     * Objects that are on it.
     */
    public MapView ()
    {
        this.getChildren().addAll(_map, _frames, _objects, _highlights);

        this.initEvents();

        //centers the map on screen.
        this.setLayoutX(Dimensions.getWidth() / 2 - this.getPxDimensions()[0]);
        this.setLayoutY(Dimensions.getHeight() / 2 - this.getPxDimensions()[1]);

        this.drawMap();
    }

    /**
     * Getter.
```

```
     *
     * @return - the MapView's dimensions in pixels for centering it on screen..
     */
    public int[] getPxDimensions ()
    {
        return _pxDimensions;
    }

    /**
     * Gets the TileView corresponding to the given tile.
     *
     * @param t - the given tile
     *
     * @return - The corresponding TileView
     */
    public TileView getTileView (Tile t)
    {
        if (this._tileMap.containsKey(t))
          {
            return this._tileMap.get(t);
          }
        return null;
    }

    /**
     * Adds an Image to a group. Image can be set to mouseTransparent or not.
     *
     * @param group            Group to add the Image to.
     * @param imageLocation    Location of the Image to add, as defined in the
     *                         resource/images package.
     * @param t                - Tile or Location of the object the Image
     *                         represents. Is used for determining the images
     *                         position
     * @param mouseTransparent - Wether or not the image is supposed to be
     *                         mouseTransparent.
     */
    private void setImage (Group group, String imageLocation, Tile t,
            Boolean mouseTransparent)
    {
        Image image = null;
        try
          {
            image = new Image(new FileInputStream(imageLocation));
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(GameScene.class.getName())
                    .log(Level.SEVERE, null, ex);
          }

        if (image != null)
          {
            setImage(group, image, t, mouseTransparent);
          }
    }

    /**
     * Adds an Image to a group. Image can be set to mouseTransparent or not.
     *
     * @param group            - Group to add the Image to.
     * @param image            - The image to add to the specified group.
     * @param t                - Tile or Location of the object the Image
     *                         represents. Is used for determining the images
     *                         position
     * @param mouseTransparent - Wether or not the image is supposed to be
     *                         mouseTransparent.
```

```
     */
    private void setImage (Group group, Image image, Tile t,
            Boolean mouseTransparent)
    {
        TileView tw = new TileView(image, t);

        tw.setX(this.getTranslateX() + t.getAxialTransform().getD() * 58 +
                t.getAxialTransform().getH() * 28);
        tw.setY(this.getTranslateY() + t.getAxialTransform().getH() * 50);
        tw.setMouseTransparent(mouseTransparent);
        group.getChildren().add(tw);
        if (group == this._map)
          {
            this._tileMap.put(t, tw);
          }
    }

    /**
     * Adds events needed to drag the Map on screen to this Map view.
     */
    private void initEvents ()
    {
        /**
         * Stores the position on a mouse click. Important for dragging the Map
         * View on screen but has no other uses.
         */
        this.setOnMousePressed(event ->
          {
            this._mouseX = event.getScreenX();
            this._mouseY = event.getScreenY();
          });

        /**
         * Allows for the map to be dragged on screen.
         */
        this.setOnMouseDragged(event ->
          {
            //calculate difference in mouse position
            this._offX = event.getScreenX() - this._mouseX;
            this._offY = event.getScreenY() - this._mouseY;
            //update mouse position
            this._mouseX = event.getScreenX();
            this._mouseY = event.getScreenY();

            //Set new position of this MapView
            this.setLayoutX(this.getLayoutX() + _offX);
            this.setLayoutY(this.getLayoutY() + _offY);

          });
    }

    /**
     * Draws everything that doesn't change during a game, namely the Tiles
     * themselves with images depending on height and obstacles.
     */
    private void drawMap ()
    {
        int[] minCoords =
          {
            0, 0
          };
        int[] maxCoords =
          {
            0, 0
          };
```

```
        for (Tile t : MapController.getInstance().getMap().getTiles().values())
          {
            /*
             * set image of the hex itself,
             * depending on height,
             * not mouseTransparent.
             */
            this.setImage(this._map, Images.HEX_HEIGHTS[t.getHeight()], t, false);

            //if obstacle present, set obstacle-image, mouseTransparent
            if ((t.getGameObject() != null) && Obstacle.class
                    .isInstance(t.getGameObject()))
              {
                this.setImage(_map, t.getGameObject().getImage(), t, true);
              }

            if (t.getAxialTransform().getD() >= maxCoords[0])
              {
                maxCoords[0] = t.getAxialTransform().getD();
              }
            if (t.getAxialTransform().getH() >= maxCoords[1])
              {
                maxCoords[1] = t.getAxialTransform().getH();
              }
          }
        this._pxDimensions[0] = (maxCoords[0] - minCoords[0]) * 58 +
                (maxCoords[0] - minCoords[0]) * 28;
        this._pxDimensions[1] = (maxCoords[0] - minCoords[0]) * 50;
        System.out.println("MapDimensions are " + this._pxDimensions[0] + " and " +
                this._pxDimensions[1]);
    }

    /**
     * Refreshes images of Highlights on a specified tile.
     *
     * @param t - Tile to refresh Highlights on.
     */
    private void refreshHighlights (Tile t)
    {
        if (t.isSelected())
          {
            this.setImage(this._highlights, Images.HEX_FRAME_SELECTED, t, true);
          }
        if (t.isHighlightedForMove())
          {
            this.setImage(this._highlights, Images.HEX_FRAME_HIGHLIGHTED_MOVE, t,
                            true);
          }
        if (t.isHighlightedForAttack())
          {
            this.setImage(this._highlights, Images.HEX_FRAME_HIGHLIGHTED_ATTACK,
                            t, true);
          }
    }

    /**
     * Refreshes images of objects on a specified tile.
     *
     * @param t - Tile to refresh objects on.
     */
    private void refreshObjects (Tile t)
    {
        if ((t.getGameObject() != null) && !Obstacle.class
                .isInstance(t.getGameObject()))
          {
            this.setImage(this._objects, t.getGameObject().getImage(), t, true);
```

```
            if (CreatureAdapter.class.isInstance(t.getGameObject()))
              {
                if (((CreatureAdapter) t.getGameObject()).getOwner() == 0)
                  {
                    this.setImage(this._frames, Images.PLAYER0_FRAME, t, true);
                  }
                else
                  {
                    this.setImage(this._frames, Images.PLAYER1_FRAME, t, true);
                  }
              }
          }
    }

    /**
     * Refreshes the MapView to adjust for changes to the map.
     */
    public void refresh ()
    {
        this._objects.getChildren().clear();
        this._frames.getChildren().clear();
        this._highlights.getChildren().clear();

        for (Tile t : MapController.getInstance().getMap().getTiles().values())
          {
            refreshObjects(t);
            refreshHighlights(t);
          }
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setLayoutY(this.getLayoutY() - heightDifference *
                (this.getLayoutY() / originalHeight));
        this.setLayoutX(this.getLayoutX() - widthDifference *
                (this.getLayoutX() / originalWidth));
    }

}
```

6.48.11 MusicAndEffect

Listing 210: MusicAndEffect.java

```
package View.UIElements;

import Resource.Constants.GeneralConstants.SoundConstants;
import java.nio.file.Paths;
```

```
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;

/**
 * The MusicAndEffect class regulates the volume of all Music and Effects, it
 * also plays and stops all music.
 */
public class MusicAndEffect
{

    /**
     * Stores the media of the menumusic.
     */
    final Media MENU_MUSIC;

    /**
     * Stors the media of the new Turn music.
     */
    final Media NEW_TURN_MUSIC;

    /**
     * Stores the media of the spawn effect music.
     */
    final Media SPAWN_EFFECT_MUSIC;

    /**
     * Stores the media of the game music.
     */
    final Media GAME_MUSIC;

    /**
     * MediaPlayer that plays the menu music.
     */
    final MediaPlayer MENU_PLAYER;

    /**
     * MediaPlayer that plays the start game music.
     */
    final MediaPlayer START_GAME_PLAYER;

    /**
     * MediaPlayer that plays the game music.
     */
    final MediaPlayer GAME_MUSIC_PLAYER;

    /**
     * MediaPlayer that plays the spawneffect music.
     */
    MediaPlayer _spawnEffectPlayer = null;

    /**
     * The amount of Volume is stored in this variable.
     */
    private double _volume = 0.0;

    /**
     * If music should be played or not.
     */
    private boolean _music = false;

    /**
     * Default Constructor initisalizes all Constants and variables.
     */
    public MusicAndEffect ()
    {
        this.MENU_MUSIC
```

```
                = new Media(Paths.get(SoundConstants.MENU_MUSIC).
                        toUri().toString());
        this.NEW_TURN_MUSIC
                = new Media(Paths.get(SoundConstants.START_NEW_TURN_MUSIC).
                        toUri().toString());
        this.SPAWN_EFFECT_MUSIC
                = new Media(Paths.get(SoundConstants.SPAWN_EFFECT).
                        toUri().toString());
        this.GAME_MUSIC
                = new Media(Paths.get(SoundConstants.ADDCHAOS_ADDLAW_MUSIC).
                        toUri().toString());
        this.MENU_PLAYER
                = new MediaPlayer(this.MENU_MUSIC);
        this.START_GAME_PLAYER
                = new MediaPlayer(this.NEW_TURN_MUSIC);
        this.GAME_MUSIC_PLAYER
                = new MediaPlayer(this.GAME_MUSIC);
        this._spawnEffectPlayer
                = new MediaPlayer(this.SPAWN_EFFECT_MUSIC);
        this._music = true;
        this._volume = 1.0;
    }

    /**
     * Getter for if music is enabled.
     *
     * @return True if music is enabled?
     */
    public boolean isMusic ()
    {
        return _music;
    }

    /**
     * Setter for if music is enabled.
     *
     * @param _music True if music should be enabled?
     */
    public void setMusic (boolean _music)
    {
        this._music = _music;
    }

    /**
     * Getter for the current Volume.
     *
     * @return - the current Volume
     */
    public double getVolume ()
    {
        return this._volume;
    }

    /**
     * Setter for the Volume.
     *
     * @param volume - the Volume as a double
     */
    public void setVolume (double volume)
    {
        this._volume = volume;
        this.MENU_PLAYER.setVolume(this._volume);
        this.START_GAME_PLAYER.setVolume(this._volume);
        this.GAME_MUSIC_PLAYER.setVolume(this._volume);
    }
```

```
    /**
     * Plays the first music that can be listened to in the game.
     */
    public void playMenuMusic ()
    {
        if (this._music)
          {
            this.MENU_PLAYER.play();
            repeatSong(this.MENU_PLAYER);
          }
    }

    /**
     * Plays the music that appears when the game is started.
     */
    public void playStartGame ()
    {
        if (this._music)
          {
            this.START_GAME_PLAYER.onEndOfMediaProperty().set(() ->
              {
                this.playGameMusic();
              });

            this.START_GAME_PLAYER.play();
          }
    }

    /**
     * The Music that will be played in the background while the game.
     */
    public void playGameMusic ()
    {
        if (this._music)
          {

            this.GAME_MUSIC_PLAYER.play();
            repeatSong(this.GAME_MUSIC_PLAYER);
          }
    }

    /**
     * Starts to play the current spawneffect.
     */
    public void playSummonSound ()
    {
        this._spawnEffectPlayer.play();
        this._spawnEffectPlayer = new MediaPlayer(this.SPAWN_EFFECT_MUSIC);
    }

    /**
     * Stops the menuMusic.
     */
    public void stopMenuMusic ()
    {
        this.MENU_PLAYER.stop();
    }

    /**
     * Stops the newTurnMusic.
     */
    public void stopStartNewTurn ()
    {
        this.START_GAME_PLAYER.stop();
    }
```

```
    /**
     * Stops the game Music.
     */
    public void pauseGameMusic ()
    {
        this.GAME_MUSIC_PLAYER.pause();
    }

    /**
     * This Method repeatS the sound of a given MediaPlayer.
     *
     * @param player - the Media Player, then we are want to repeat
     */
    private void repeatSong (MediaPlayer player)
    {
        player.setAutoPlay(true);
        player.setCycleCount(MediaPlayer.INDEFINITE);
    }

    /**
     * Stops all Musics.
     */
    public void stopAll ()
    {
        this.MENU_PLAYER.stop();
        this.GAME_MUSIC_PLAYER.stop();
        this._spawnEffectPlayer.stop();
        this.START_GAME_PLAYER.stop();
    }

}
```

6.48.12 PlayerWandView

Listing 211: PlayerWandView.java

```
package View.UIElements;

import Model.Card.Wand;
import Resource.Dimensions;
import Resource.VideoOptions;
import javafx.geometry.Pos;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;

/**
 * The PlayerWandView exists in the Setup Scene and shows which wand has
 * currently been selected for the player.
 */
public class PlayerWandView extends StackPane implements IChaosUI
{
    /**
     *
     */
    private int _player = 0;

    /**
     *
     */
    private ImageView _wandImage = null;

    /**
     *
     */
```

```
    private TextWithBackground _text = null;

    /**
     * The PlayerWandView exists in the Setup Scene and shows which wand has
     * currently been selected for the player.
     */
    public PlayerWandView ()
    {
        this._text = new TextWithBackground();
    }

    /**
     * The PlayerWandView exists in the Setup Scene and shows which wand has
     * currently been selected for the player.
     *
     * @param i - The ownerID of the corresponding player.
     */
    public PlayerWandView (int i)
    {
        this._text = new TextWithBackground();
        this._player = i;
        this.setMouseTransparent(true);
        this.init();
    }

    /**
     * Sets all default values.
     */
    private void init ()
    {
        this.setAlignment(Pos.CENTER_LEFT);
        this.setMinHeight(500);

        this._text.setText("Player " + this._player + ":", 24);

        this.getChildren().add(this._text);
    }

    /**
     * Sets the Wand this WandView is supposed to show.
     *
     * @param w - the corresponding Wand.
     */
    public void setWand (Wand w)
    {
        this._wandImage = new ImageView(w.getImage());
        this._wandImage.setScaleX(0.5d);
        this._wandImage.setScaleY(0.5d);
        this._wandImage.setTranslateX(80);
        this._wandImage.setEffect(new DropShadow(20, Color.WHITESMOKE));
        this.getChildren().add(this._wandImage);
    }

    /**
     * Getter.
     *
     * @return - True if this View is showing a Wand.
     */
    public boolean hasWand ()
    {
        if (this._wandImage != null)
          {
            return true;
          }
        return false;
    }
```

```
    /**
     * Getter
     *
     * @return - the currently selected wand.
     */
    public PlayerWandView get ()
    {
        return this;
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference
                * (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference
                * (this.getTranslateX() / originalWidth));
    }
}
```

6.48.13 RoundButton

Listing 212: RoundButton.java

```
package View.UIElements;

import Resource.Dimensions;
import Resource.Images.Images;
import Resource.VideoOptions;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.geometry.Pos;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;

/**
 * A Round Button is a Round Button with some nice Visuals.
 */
public class RoundButton extends StackPane implements IChaosUI
{
```

```
    /**
     *
     */
    private Text _text = null;

    /**
     * This Buttons hitbox.
     */
    private Circle _clickBox = null;

    /**
     *
     */
    private String[] _images =
      {
        Images.END_TURN_IDLE, Images.END_TURN_HOVER, Images.END_TURN_PRESSED
      };

    private ImageView _bg = null;

    /**
     * A Round Button is a Round Button with some nice Visuals.
     */
    public RoundButton ()
    {
        this._text = new Text();
        this._clickBox = new Circle(64);
        this.changeIcon(0);
        this.init();
    }

    /**
     * A Round Button is a Round Button with some nice Visuals.
     *
     * @param title - The Text displayed on this Button.
     */
    public RoundButton (String title)
    {
        this._text = new Text();
        this._clickBox = new Circle(64);
        this._text.setText(title);
        this.changeIcon(0);
        this.init();
    }

    /**
     * Sets all Default Values.
     */
    private void init ()
    {
        this.setScaleX(1.5);
        this.setScaleY(1.5);

        this.setTranslateY(VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT /
                2 - 64);
        this.setTranslateX(VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH - 128 *
                1.5d - 50);

        this.setAlignment(Pos.CENTER);

        this.initText();
        this.initHitBox();
        this.setMouseEvents();
    }

    /**
```

```
     * Adds the HitBox to this Button.
     */
    private void initHitBox ()
    {
        this._clickBox.setFill(Color.TRANSPARENT);
        this.getChildren().add(this._clickBox);
    }

    /**
     * Adds Text to this Button.
     */
    private void initText ()
    {
        this._text.setFont(Font.font("Book Antiqua", FontWeight.BOLD, 20));
        this._text.setMouseTransparent(true);
        this._text.setEffect(new DropShadow(5, Color.GHOSTWHITE));
        this._text.setFill(Color.WHITE);
        this._text.toFront();

        this.getChildren().add(this._text);
    }

    /**
     * Sets The mouseEvents for this button relevant for changing the background
     * image.
     */
    private void setMouseEvents ()
    {
        /**
         * Mouse entered event: moves shape to right and swaps background and
         * text color to indicacte which button the mouse is on (for fancy
         * looks)
         */
        this.setOnMouseEntered(event ->
          {
            this.changeIcon(1);
            _text.setFill(Color.BLACK);
          });

        /**
         * Mouse exited event: revert changes made on mouse entered.
         */
        this.setOnMouseExited(event ->
          {
            this.changeIcon(0);
            _text.setFill(Color.WHITE);
          });

        //...which is  used in  the onMousePressed event...
        this.setOnMousePressed(event ->
          {
            this.changeIcon(2);
            _text.setFill(Color.BLACK);

          });

        this.setOnMouseReleased(event ->
          {
            this.onMouseReleased();
          });
    }

    /**
     * Changes the background image to the icon corresponding to its internal
     * ID.
     *
```

```
     * @param i the ID of the icon
     */
    private void changeIcon (int i)
    {
        this.getChildren().remove(_bg);
        try
          {
            this._bg = new ImageView(new Image(new
        FileInputStream(this._images[i])));
            this._bg.setMouseTransparent(true);
          }
        catch (FileNotFoundException ex)
          {
            Logger.getLogger(RoundButton.class.getName())
                    .log(Level.SEVERE, null, ex);
          }

        this.getChildren().add(_bg);
        _text.toFront();
    }

    /**
     * Extra Method for onMouseReleased, so that it may be called from another
     * class.
     */
    public void onMouseReleased ()
    {
        this.changeIcon(1);
        _text.setFill(Color.BLACK);
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference *
                (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference *
                (this.getTranslateX() / originalWidth));
    }
}
```

6.48.14 TextWithBackground

Listing 213: TextWithBackground.java

```
package View.UIElements;

import Resource.Dimensions;
import Resource.VideoOptions;
import javafx.geometry.Pos;
```

```
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;

/**
 * Our Implementation of a Text, which has a background that is automatically
 * sized to fit the text.
 */
public class TextWithBackground extends StackPane implements IChaosUI
{

    /**
     *
     */
    private Rectangle _bg = null;

    /**
     *
     */
    private Text _text = null;

    /**
     *
     */
    private int _fontSize = 0;

    /**
     * Our Implementation of a Text, which has a background that is
     * automatically sized to fit the text.
     */
    public TextWithBackground ()
    {
        this._bg = new Rectangle();
        this._text = new Text();
        this.init();
    }

    /**
     * Our Implementation of a Text, which has a background that is
     * automatically sized to fit the text.
     *
     * @param s         - The text that will be displayed
     * @param fontSize - The desired font size.
     */
    public TextWithBackground (String s, int fontSize)
    {
        this._bg = new Rectangle();
        this._text = new Text();

        this._fontSize = fontSize;
        this.setText(s);

        this._text.setFont(Font.font("Book Antiqua", FontWeight.BOLD,
                                                                this._fontSize));

        this.init();
    }

    /**
     * Getter.
     *
     * @return The Width of the background
```

```
     */
    public int getTotalWidth ()
    {
        return (int) this._bg.getWidth();
    }

    /**
     * Sets a new Text for this textfield.
     *
     * @param s The new Text.
     */
    public void setText (String s)
    {
        this._text.setText(s);
        this._text.setTranslateX(this._fontSize / 2);

        this._bg.setHeight(this._fontSize + 6);
        this._bg.setWidth(s.length() * this._fontSize / 2 + 20);
    }

    /**
     * Sets a new Text and fontSize for this textfield.
     *
     * @param s        The new Text.
     * @param fontSize The new fontSize.
     */
    public void setText (String s, int fontSize)
    {
        this._fontSize = fontSize;
        this._text.setFont(Font.font(this._text.getFont().toString(),
                                     FontWeight.BOLD, this._fontSize));
        this.setText(s);
    }

    /**
     * Sets all relevant default values.
     */
    private void init ()
    {
        this._bg.setOpacity(0.6d);

        this.setEffect(new DropShadow(10, Color.WHITESMOKE));
        this.setAlignment(Pos.CENTER_LEFT);

        this._text.setFill(Color.WHITE);

        this.getChildren().add(this._bg);
        this.getChildren().add(this._text);
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor. Inherited
     * from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;
```

```
        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference *
                (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference *
                (this.getTranslateX() / originalWidth));
    }

}
```

6.48.15 TileView

Listing 214: TileView.java

```
package View.UIElements;

import Control.Controllers.SceneController;
import Control.SemiControllers.Selector;
import Model.Map.Tile;
import View.Tooltip.TileTooltip;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseButton;

/**
 * The Tile view visually represents a single _tile on the map and the Object on
 * it.
 */
public class TileView extends ImageView
{

    /**
     * Tile this Tile View represents on screen.
     */
    private Tile _tile = null;

    /**
     * Tooltip that is shown when this _tile is selected.
     */
    private TileTooltip _tooltip = null;

    /**
     * The Tile view visually represents a single _tile on the map and the
     * Object
     * on it.
     *
     * @param im This Tile's Image.
     * @param t  Tile this TileView represents on screen.
     */
    public TileView (Image im, Tile t)
    {
        super(im);
        this._tooltip = new TileTooltip(t);
        this._tile = t;

        this.setOnMouseClicked(event ->
          {
            System.out.println(this._tile.toString());
            this._tooltip.show();

            if (event.getButton() == MouseButton.PRIMARY)
              {
                Selector.getInstance().selectTile(this._tile);
              }
            if (event.getButton() == MouseButton.SECONDARY)
```

```
                {
                  Selector.getInstance().selectSecTile(this._tile);
                  Selector.getInstance().deselectTile();
                }

              SceneController.getInstance().refresh();
            });
      }

      /**
       * Getter.
       *
       * @return - Tile this TileView represents.
       */
      public Tile getTile ()
      {
          return this._tile;
      }
}
```

6.48.16 TurnPlayerDisplay

Listing 215: TurnPlayerDisplay.java

```
package View.UIElements;

import Control.Controllers.GameController;
import Resource.Constants.GeneralConstants.UIConstants;
import Resource.Dimensions;
import Resource.VideoOptions;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.GaussianBlur;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;

/**
 * Text that displays which player's turn it currently is.
 */
public class TurnPlayerDisplay extends StackPane implements IChaosUI
{

    /**
     *
     */
    private Text _text = null;

    /**
     *
     */
    private Rectangle _bg = null;

    /**
     * Text that displays which player's turn it currently is.
     */
    public TurnPlayerDisplay ()
    {
        this._text = new Text();
        this._bg = new Rectangle(250, 30);
        this.init();
    }

    /**
     * Sets all default values.
     */
```

```
    private void init ()
    {
        this.setScaleX(2.5d);
        this.setScaleY(2.5d);

        this.setTranslateX(UIConstants.TURN_PLAYER_DISPLAY_X);
        this.setTranslateY(UIConstants.TURN_PLAYER_DISPLAY_Y);

        this.initBG();
        this.initText();

        this.refresh();
    }

    /**
     * sets the Background for this Element.
     */
    private void initBG ()
    {
        this._bg.setOpacity(0.8);
        this._bg.setFill(Color.LIGHTGRAY);

        this._bg.setEffect(new GaussianBlur(2));
        this.getChildren().add(this._bg);
    }

    /**
     * sets the Text on this Element.
     */
    private void initText ()
    {
        this._text.setFont(_text.getFont().font(20));
        this.getChildren().add(this._text);
    }

    /**
     * Refreshes the current player's number and selects a text color
     * accordingly.
     */
    public void refresh ()
    {
        this._text.setText("Player " +
    (GameController.getInstance().getCurrentTurnController().getCurrentPlayer()
        .getOwnerNumber()
            + 1) + "'s Turn");

        switch
    (GameController.getInstance().getCurrentTurnController().getCurrentPlayer()
        .getOwnerNumber())
          {
            case 0:
                this._text.setFill(Color.WHITE);
                this._bg.setFill(Color.FIREBRICK);
                this._bg.setStroke(Color.WHITE);
                this._text.setEffect(new DropShadow(2, Color.WHITE));
                break;
            case 1:
                this._text.setFill(Color.BLACK);
                this._bg.setFill(Color.YELLOW);
                this._bg.setStroke(Color.BLACK);
                this._text.setEffect(new DropShadow(2, Color.BLACK));
                break;
            default:
                this._text.setFill(Color.WHITE);
                break;
          }
```

```
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference *
                (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference *
                (this.getTranslateX() / originalWidth));
    }

}
```

6.48.17 WandView

Listing 216: WandView.java

```
package View.UIElements;

import Model.Card.Wand;
import Model.Card.WandFactory;
import Resource.Dimensions;
import Resource.VideoOptions;
import javafx.geometry.Pos;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.Glow;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;

/**
 * Each WandView represents one of the selectable Wands in the SetupMenu.
 */
public class WandView extends StackPane implements IChaosUI
{
    /**
     *
     */
    private int _wandID = 0;

    /**
     *
     */
    private Wand _wand = null;

    /**
     *
     */
    private ImageView _wandIm = null;
```

```
    /**
     * Each WandView represents one of the selectable Wands in the SetupMenu.
     *
     * @param id - The ID of the corresponding Wand.
     */
    public WandView (int id)
    {
        this._wandID = id;
        this._wand = WandFactory.getWand(id);
        this.init();
    }

    /**
     * Getter.
     *
     * @return The Wand corresponding to this WandView.
     */
    public Wand getWand ()
    {
        return WandFactory.getWand(this._wandID);
    }

    /**
     * Sets all default Values.
     */
    private void init ()
    {
        this._wandIm = new ImageView(WandFactory.getWand(this._wandID).getImage());

        this.setAlignment(Pos.CENTER);

        this.setMouseEvents();

        this.setEffect(new DropShadow(20, Color.WHITESMOKE));

        this.getChildren().add(this._wandIm);
    }

    /**
     * These Mouse Events visually emphasize which wand the player is currently
     * hovering over and selecting.
     */
    private void setMouseEvents ()
    {
        this.setOnMouseEntered(event ->
          {
            this._wandIm.setEffect(new Glow(10));
            this._wandIm.setTranslateY(-20);
            this._wand.showTooltip();
          });

        this.setOnMouseExited(event ->
          {
            this._wandIm.setEffect(null);
            this._wandIm.setTranslateY(20);
            this._wand.hideTooltip();
          });
    }

    /**
     * Scales this UIElement to fit the Size of the active monitor.
     * Inherited from IChaosUI.
     */
    @Override
    public void scaleToMonitorSize ()
    {
```

```
        double monitorHeight = Dimensions.getHeight();
        double monitorWidth = Dimensions.getWidth();

        double originalHeight = VideoOptions.DEFAULT_RENDER_RESOLUTION_HEIGHT;
        double originalWidth = VideoOptions.DEFAULT_RENDER_RESOLUTION_WIDTH;

        double heightDifference = originalHeight - monitorHeight;
        double widthDifference = originalWidth - monitorWidth;

        this.setScaleY(this.getScaleY() * (monitorHeight / originalHeight));
        this.setScaleX(this.getScaleX() * (monitorWidth / originalWidth));

        this.setTranslateY(this.getTranslateY() - heightDifference *
                (this.getTranslateY() / originalHeight));
        this.setTranslateX(this.getTranslateX() - widthDifference *
                (this.getTranslateX() / originalWidth));
    }
}
```

7 Art

7.1 AirElemental

7.2 AirWand

7.3 BlueExplosion

7.4 Buff

7.5 CreatureWand

7.6 Death

7.7 Deck

7.8 Decree

7.9 DivineBow

7.10 DivineShield

7.11 DivineSword

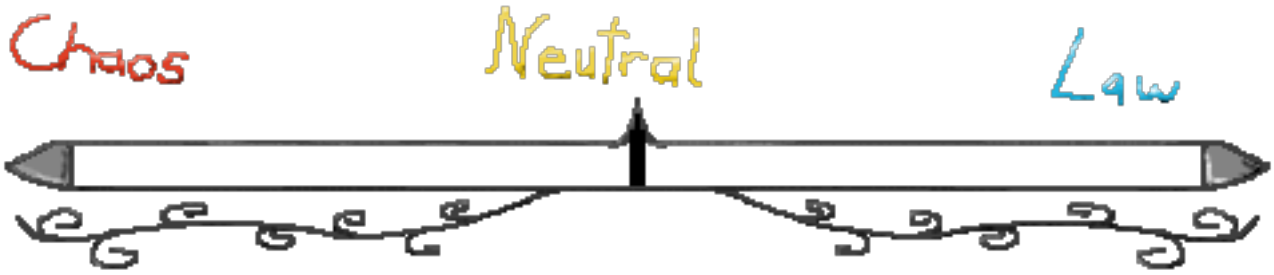
7.12 Eagle

7.13 EndTurnHover

7.14 EndTurnIdle

7.15 EndTurnPressed

7.16 Flux



7.17 Frame_Player_Lime

7.18 Frame_Player_Red

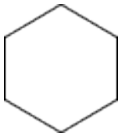
7.19 Frame_Player_Yellow

7.20 GameBackground

7.21 Giant

7.22 Hellhound

7.23 Hex_Frame

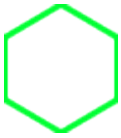


7.24 Hex_Frame_Highlighted_atk

7.25 Hex_Frame_Highlighted_atk_move

7.26 Hex_Frame_Highlighted_move

7.27 Hex_Frame_Selected



7.28 Hex_Height_0

7.29 Hex_Height_1

7.30 Hex_Height_2

7.31 Hex_Height_3

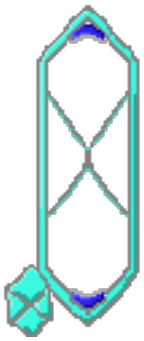
7.32 Hit

7.33 Hydra

7.34 IcarusTower

7.35 MagicBolt

7.36 ManaBar



7.37 ManaCost

7.38 Mana_Crystal

7.39 Manticore

7.40 MenuBackground0

7.41 MenuBackground1

7.42 MenuBackground2

7.43 MenuBackground3

7.44 NoImage

7.45 Obstacle

7.46 Paladin

7.47 Pegasus

7.48 PlayerCharacter_Wizard_lime

7.49 PlayerCharacter_Wizard_purple

7.50 PlayerCharacter_Wizard_red

7.51 PlayerCharacter_Wizard_yellow

7.52 RandomWand

7.53 SapphireDragon

7.54 Skeleton

7.55 Summon

7.56 Lion

7.57 UndeadWand

7.58 Vampire

7.59 Vengeance

7.60 Windwalker

7.61 Zombie