Projektplan des Teams Gruppe 1

Team “Boom Softwares”

Spiel “Boom Chess”

Programmierprojekt Informatik 2
WS 23/24

Lauterbach Martin
Cam Minh Dan
Yasheem Bashar
Ruckgaber Raphael

Logo erstellt mit DalleE3 powered by GPT4, 16.10.23

Ausgewählte Pakete:

Basis Backend, Basis Frontend, Frontend erweitert

Inhaltsangabe

1. Die allgemeine Spielidee

Ein “modernes” Schachspiel.

Anstatt den normalen Figuren, ist das 9x8 große Feld durch moderne Militärfiguren ersetzt.
Der König ist ein General.
Der Turm ein Panzer.
Der Läufer sind Wachhunde.
Die Bauern sind Infanterie.
Der Reiter ist ein Helikopter.
Eine zusätzliche Spielfigur ist die Artillerie - 2 Felder weit nutzbar.

Anstatt das die Figuren sich schlagen, machen Sie jeweils in ihrem Umfeld befindlichen feindlichen Figuren Schaden, der an der Art des Angreifers und Verteidigers abhängt.

Die Figuren haben dementsprechend eine Lebenspunkteanzahl und machen einen zufälligen Schaden innerhalb einer min-max-Spanne. Diese Spieldynamik entfernt das Spielkonzept des Schlagens, und fügt einen auf Zufall und Abwägung basiertes Konzept ein.

Das Spiel wird designmäßig in einer orthographischen Sicht gestaltet und als Artstyle Wird Pixelart für die Spielfiguren und Illustrationen für die Spielkarte genutzt.
Die Farben des Spiels sollen gedämpft und in das Setting einer Simulation einer Militärsimulation passen - der Hintergrund ist der Schreibtisch eines Generals selbst, mitsamt markanten Objekten wie einem Skizzenblatt, einem Kaffee, Notizblock etc.

Man kann die Karte auswählen, auf dem zwei Generäle das Boom-(Schach)Chess-Spiel spielen.

Der Spieler soll ein Feedback, wer wen angreift, über rot eingezeichnete Linien zwischen Figuren bekommen. Weiter soll es noch Geräusche, Musik und bei 0 Leben eine Explosions-Animation geben.

So weit wie machbar, sollen Spielfiguren Animationen erhalten.

Das Spiel soll entweder zu zweit lokal oder lokal gegen einen Bot gespielt werden können.
Spielmodi 1.

Bestimmte Spielattribute wie die Farbe der Uniformen der Spielfiguren soll zur Inklusivität bei Farbenblindheit veränderbar sein. Die Grünen Uniformen sollen zu Blau gewechselt werden.
Darüber hinaus soll das Militär-Setting zu einem Mittelalter-Setting geändert werden können - Spielmodus 2.

Die Spielfiguren sollen über Drag und Drop bewegt werden können, wenn es ihr Zug ist. Während des Drag sollen mögliche Tiles, auf denen Sie landen können, markiert werden.
Es soll die Möglichkeit geben, auf dem Spielfeld zufällig Blockaden generieren zu lassen, auf denen Schachfiguren nicht draufkommen können. Dies soll als dritter Spielmodus spielbar sein.

Für das Memory-Management sollen Texturen am Anfang des Spiels geladen werden und dann in Image-Objekte umgewandelt werden, die bei ReRendering des Spielbrett durch Java garbaged werden können.

2. Ausgewählte Pakete vorgestellt

Das Projekt wird aus den folgenden Paketen erstellt werden:

Basis Backend

Das Basis Backend soll im Projekt gesondert im Directory Backend ansässig sein.
Dort werden eine abstract class Soldier, die subklassen der Spielfiguren sowie die Klassen von Schlüsselspielfunktionen gelagert, die dann während des Main Game Loops in BoomChess.Java genutzt werden sollen.

Game-Loop Controller (5P)

Ein Main Game Loop wird in BoomChess.java gerichtet an dem späteren Rendern jedes Frame durchlaufen. Bestimmte Parameter wie, ob das Spiel gerade (boolean inGame) in einer Runde ist oder welcher Spieler dran ist (Game State) sollen die Eingabe auf bestimmte Schachfiguren limitieren. Sobald ein Spieler einen erlaubten Zug macht, soll der Game-Loop dann bestimmte Funktionen wie die Schadensermittlung (Damage.class) durchlaufen und die Spielparameter auf den nächsten Zug ändern.

Der Game-Loop-Controller wird Teil der immer wachsenden BoomChess.java Klasse sein. Wir schätzen daher ein, dass sich die Dauer der Entwicklung ziehen wird.

Spielfeld (5P)

Das Spielfeld ist ein in der Board.java class erstelltes 2D-Array an Soldier Objekten (den Schachfiguren). Ihre Bewegung, Schaden, Team, Art und Aussehen werden im Regelwerk festgelegt und verändern ihre Funktionalität.
Das Spielfeld kann verändert werden, indem sich Spielfiguren bewegen (ein Tauschen der Objekte zwischen Punkt A und Punkt B) oder eine Spielfigur stirbt (killPiece in Damage.java).

Die Erstellung eines 2D-Array wird vergleichbar kurz auf 2 Stunden gesetzt von Uns, während wir bei der Erstellung der Soldier-Objekten mit einem Zeitaufwand von bis zu 20 Stunden rechnen werden. Die Funktionalitäten müssen in Ihren Sonderformen, dass wir viele verschiedene Arten an Spielfiguren und Eigenschaften planen, resistent gegen Exceptions und Bugs gemacht werden.

Spieler (2P)

Der Spieler selbst ist repräsentiert durch die Farben Rot und Grün. Ein Zug des jeweiligen Spielers wird durch die Restriktion der Bewegung des anderen Spielers verfestigt.
Der grüne Spieler ist auf der linken Seite des Feldes, während Rot auf der rechten Seite des Feldes ist.

Die Erstellung des Spielers erscheint Uns schnell gemacht zu sein. Dieser wird durch eine reine Limitierung imitiert, die wir durch einen enum simulieren möchten.
Wir erwarten einen Zeitaufwand von bis zu 2 Stunden.

Regelwerk (5P)

Das Regelwerk soll die Besonderheiten in der Interaktion zwischen den Spielfiguren und die eigenen Funktionalitäten widerspiegeln. Wir planen, dass beispielhaft eine Helikopter-Spielfigur nur auf Felder, mit einer Entfernung x -+ 3, y +- 1 laufen kann, Schaden auf einer Entfernung von einem Feld machen, wenig Schaden von einem Panzer nehmen soll aber dafür mehr Schaden außerhalb ihres 01-20 Schadens auf Infanterie mit einem Bonus von +5 zu machen.
Diese Besonderheit soll durch den einzelnen Subclassen hinzugehörenden Methods realisiert werden, die jeweils mögliche Bewegungen, Schaden anhand des Verteidigers und Schaden anhand des Angreifers berechnen sollen.

Wir rechnen bei einer Implementierung aller Eigenschaften bei allen Spielfiguren mit einem Zeitaufwand von bis zu 10 Stunden.

Konsolen-Frontend inkl. Interface (3P)

Das Konsolen-Frontend inkl. Interface planen wir auf einer von zwei Weisen zu realisieren.

In der ersten Art und Weise soll das Schachbrett auf das Konsolenfenster Reihe an Reihe anhand des 2D-Array (Board.java) gezeichnet werden. Zudem sollen wichtige Informationen wie Schadens- und Lebensmeldungen angezeigt werden.
Daraufhin wird eine Eingabe des Spielers z.B:. durch a5, b6 erwartet. Ist diese Bewegung möglich, wird Sie ausgeführt, und der nächste Spieler ist am Zug.

In der zweiten Art und Weise soll ein grundlegendes Frontend diese Darstellung durch eine vorgreifende Nutung von libGDX mit Scene2DUI diese bereits hier übernehmen.

Wir planen mit einem Zeitaufwand von bis zu 5 Stunden.

Basis Frontend

• Spielsetup Phase/Ansicht (3P)

Das Spielsetup des Frontends soll durch eine grafische Renderung von sogenannten Stages geschehen. Stages sind Objekte von Scene2DUi, einem libGDX angehörigen Packages, das die strukturierte Darstellung von Texturen, Text, Knöpfen, Kontrollleisten etc. kontrollierter programmieren lassen soll.

Das Spiel soll in ein Hauptmenü, erstellt durch die MainMenuStage (class) geladen werden. Hier sollen beispielhaft das Logo sowie bestimmte funktionelle Knöpfe es ermöglichen, das Spiel zu steuern. Es sollen die verschiedenen Spielmodi erstellt werden, sowie das Regelwerk angezeigt und die Optionen verändert werden können.

Wir rechnen mit dem Erlernen der Objekte und speziellen Funktionen von Scene2DUI mit einem Aufwand von bis zu 20 Stunden.

• Darstellung des Spielfelds (5P)

Das Spielfeld soll durch die sogenannte GameStage dargestellt werden. Sie wird gerendert, sobald der Spieler das Spiel startet. SIe rendert zentriert auf dem Bildschirm die verschiedenen Schachfiguren auf einer separaten MapStage, welche die derzeit ausgewählte Karte unter dem Spielfeld rendert.
Das Schachbrett erstellt sich, in dem es durch das 2D-Array von Board.java durchgeht und jedes einzelne Objekt nach seiner Textur (Interface takeSelfie) fragt.

Wir rechnen mit einem Aufwand von 10 Stunden.

• Anzeige vom Spielerstatus (2P)

Der Spielerstatus soll durch eine Darstellung von Lebensbalken geschehen, die den aktuellen Stand der health variable der jeweiligen Schachfigur hergibt.

Zudem soll dargestellt werden, wer gerade am Zug ist, durch ein Motiv auf dem Spielfeld das entweder angibt “Rot ist am Zug” oder “Grün ist am Zug”.

Wir rechnen mit einem Aufwand von 2 Stunden.

• Interaktion um einen Zug durchzuführen (5P)

Um eine Interaktion des Spielers mit dem Spielfeld zu ermöglichen, wollen wir die Funktionalität von Scene2DUI nutzen, und den einzelnen Texturen der GameStage es erlauben, auf einen bestimmten Punkt gezogen zu werden, der einen Befehl widerspiegelt, diese Figur an diesen Punkt auf dem 2D-Array des Backends zu bewegen.

Der Spieler soll nur erlaubt sein, diese Bewegung zu vollenden, wenn sich die Spielfigur auf einer erlaubten Position befindet, die durch die Bewegungseigenschaft des Soldierobjekts dirigiert wird.

Wir rechnen mit einem Aufwand von bis zu 10 Stunden, wegen den technischen Implikationen der Schnittstelle Frontend Pixel zu Backend 2D-Array Index.

• Anzeige für Spielende + neue Runde starten (2P)

Wenn das Spiel endet, und das soll geschehen, wenn ein General stirbt, soll eine neue Stage über der GameStage gelagert werden, welche den Gewinner jubiliert und die Möglichkeit eines Neustarts oder Rückkehr in das Hauptmenü durch Scene2DUI Knöpfe, kombiniert durch ein neuladen der GameStage, ergeben soll.

Wir rechnen mit einem Aufwand von einer Stunde.

• Anbindung an das Interface des Backends (3P)

Das Frontend ist an das Backend durch das gemeinsame Arbeit an dem 2D-Array dargestellt. Während das Frontend dieses darstellt und Befehle des Spielers bearbeitet, muss das Backend Befehle verifizieren (wie die möglichen Bewegungen) und Befehle ausführen (wie das Verändern eines Objekts von Index A nach Index B, oder eines "löschen" eines Objekts durch das Ersetzen durch ein Soldier Empty Objekts).

Diese Funktionalitäten sollen durch Schnittstellen von Funktionen zu den Soldier subClassen erreicht werden. Das Frontend soll durch eine takeASelfie Interface ein Texturen Objekt das zuvor in BoomChess.java geladen wurde erhalten.

Wir rechnen mit einer Aufwand von bis zu 4 Stunden.

Frontend erweitert

- Drag&Drop Feature (5P)

Das Drag und Drop Feature wollen wir durch sogenannte Drag Active Listeners aus libGDX erreichen. Dazu fügen wir jedem Textur-Objekt aus GameStage, welches jeweils eine Schachfigur darstellt, diese Active Listeners hinzu. Dieses ermöglichen es, jedes Frame die neue Position der Textur auf der Stage (dem Bildschirm) zu ermitteln und an diesem Ort zu rendern. Wenn ein Drop (loslassen der Maus) passiert, soll der letzte X und Y Wert von uns aufgenommen und gemeinsam mit den Eigenschaften des Bildschirms (Höhe und Breite), sowie den Eigenschaften des zentrierten Spielfelds, zu einem Index Wert des 2D Array übersetzt / kalkuliert werden.
Dieser IndexWert soll abgeglichen werden mit den zu Beginn des Drags durch die Soldier subclass method calculateMoves ausgerechneten möglichen Bewegungen.
Wenn dieser übereinstimmt, soll das Backend den Befehl UpdatePosition erhalten.

Zu Beginn des Drags soll die Frontend erweitert: Anzeige möglicher Züge ausgelöst werden.

Wir rechnen von einem Aufwand von bis zu 10 Stunden.

- Aufwändigere/s Design/Assets (5P)

Wir planen das Design in einem bereits erwähnten generals-schreibtischartigen Konstrukt darzustellen.
Dafür sollen über die Nutzung von Bildbearbeitungsprogrammen entweder selbst oder (gekennzeichnet und benannt) von AI erstellte Assets erweitert und zusammengefügt werden und als Texturen global im Spiel geladen am Anfang in BoomChess.java in Scene2DUI genutzt werden.

Wir planen Texturen, Animation und Geräusche in einer Masse in speziellen Objekten zu kategorisieren, die die Möglichkeit und Methoden haben, zufällige Texturen herauszugeben oder abzuspielen. Dies soll die Vielseitigkeit des Spiels hervorheben und dem Spieler Spaß machen.

Passen zu den einzelnen Spielfiguren sollen diese jeweils selbst speziell kategorisierte Geräusche bei Angriffen von sich geben können. Ein Interface zwischen Frontend und Soldaten, dem Backend, soll diese abspielen können.

Wir rechnen von einem Zeitaufwand von bis zu 30 Stunden.

- Aufwändigere Menü/Spielstruktur (5P)

Das Spiel soll durch mehr Funktionalitäten wie steuerbarer Hintergrundmusik, Geräuschpegel von Angriffsgeräuschen, einem indiviuellem Abhandeln von Angriffen etc. erweitert werden.

Ein sogenannter AudioTable soll jederzeit auf dem Bildschirm verfügbar sein und es ermöglichen, alle audiobezogene Dinge des Spiels zu steuern.

Die Angriffe sollen als gestrichelte Linien zwischen dem Angreifer und dem Verteidiger dargestellt werden. Diese Linien sollen je nach Team eine eigene Farbe haben. Der Verteidiger soll jeweils eine kleine Explosions-Animation (HitMarker) erhalten.
Bei einem Tod soll eine große Explosion erscheinen.

All diese “Actors” (Term von Scene2DUI, stellen Objekte auf einer Stage dar) sollen nach jedem Zug gesammelt werden und dann in einer actionSequence (Objekt) abgespielt werden, welches ein Fortfahren des Spiels vor Ablauf aller Sequenzen verhindert.
Das soll dem Spieler die Möglichkeit geben, das Gefühl der Immersion zu vertiefen und mehr in das Kampfgeschehen einzutauchen.

Wir rechnen mit einem Aufwand von bis zu 20 Stunden.

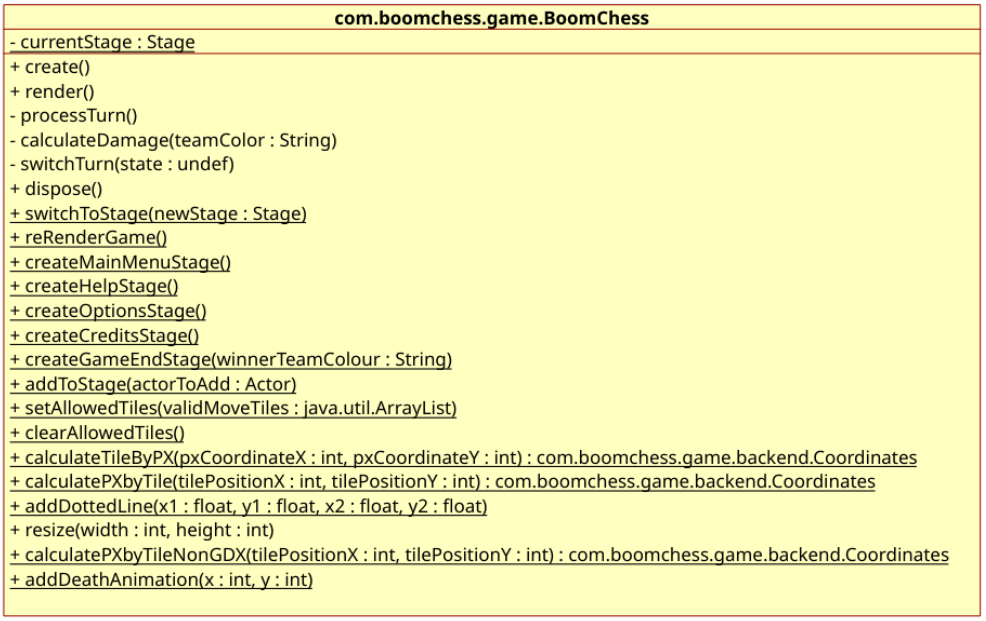
- Anzeige möglicher Züge (5P)

Mögliche Züge sollen visuell dargestellt werden, indem bei Start eines Dragings einer Spielfigur deren mögliche Züge ausgerechnet werden. Danach wird das Spielbrett neu geladen und bei Durchlaufen des Frontends durch das 2D Array gleicht dieses alle Koordinaten im Loop mit allen möglichen Bewegungskoordination ab. Wenn es ein Match gibt, soll die Schachbrettposition ein markantes Zeichen erhalten, das dem Spieler symbolisiert, dass es sich dorthin bewegen kann. Wenn ein Drag endet, also ein Drop passiert ist, soll diese Liste gelöscht werden und das Schachbrett neu geladen.

Wir rechnen für eine Implementierung mit einem Aufwand von 5 Stunden, da das Konzept in bestehende Algorithmen implementiert wird.

3. Modulerklärungen und Prognosen

3.1 Main-Class BoomChess



Funktion:
Die BoomChess.Java Datei ist die Main file des Spiels. Im Framework von libGDX soll Sie über Gradle aufgerufen werden - bzw überschreibt Sie bestimmte Methoden des Frameworks:
- create - render - dispose.

Aufbau:
Create wird als erste Method aufgerufen - sie initialisiert Variablen, lädt Dateien ins Projekt und startet die erste Stage, die der User auf seinem Bildschirm sieht und mit der er interagieren kann.

Render ist die Main-Game loop. Sie läuft unentwegt. Wir bezeichnen die derzeit darzustellende Stage in der Variable currentStage. Bestimmte Stages, wie die Audiokonfiguration, sollen außerhalb der currentStage-Variable und geben Ihr einen InputProcessor, damit mit Ihr interagiert werden kann.
In Render ist zudem die processTurn method - wenn wir im Spiel sind, was durch das enum !Not_in_Game dargestellt werden soll, wird je nachdem welche Farbe im Zug ist solange das Spiel ohne Veränderungen gerendert, bis der Spieler einen legitimen Zug gemacht hat. Dann wird die Farbe am Zug durch eine Enum Wechsel gewechselt.

Es gibt für jede Stage im Spiel eine method, die diese Stage class erstellt. Diese Funktion gibt jeweils diese Stage zurück, was dazu führt, das wir mit diesen Methods gemeinsam die switchToStage method nutzen.

Die switchToStage method nimmt ein Stage object, leert das derzeitige currentStage, und setzt die über den Parameter erhaltene Stage als currentStage.

In BoomChess festgelegt sind Methoden zur Funktionalität des “möglichen Zug”-Overlays. Eine Stage, die sich auf die currentStage legt, wenn gerade eine Figur im Drag ist, was eine boolean Variable für die Render-Methode flippt.

Die restlichen Methoden kümmern sich um die Kalkulation von Schachbrett-Tiles zu Pixel und Pixel zu Schachbrett-Tiles. SIe funktionieren, indem der linke untere Punkt des Schachbretts errechnet wird und daran dann die Koordinaten * die Breite/Höhe der Kachel + Besonderheiten nach Funktion kalkuliert wird.

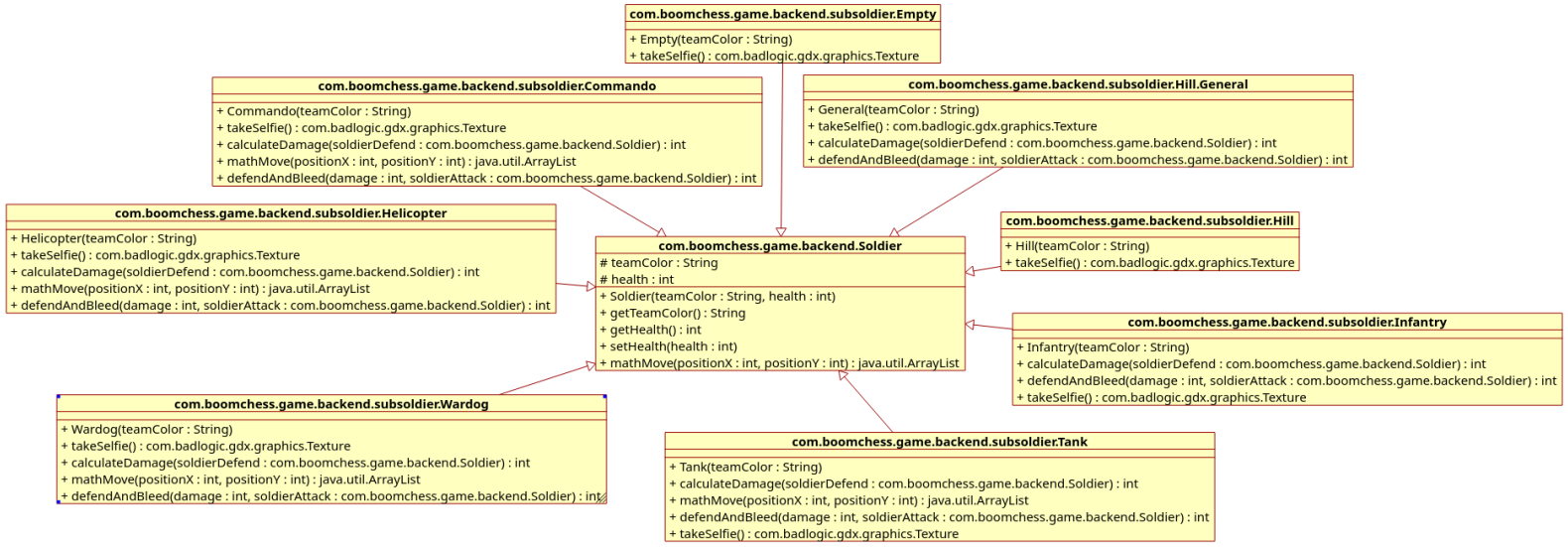
Position im Program:
BoomChess.Java ist die am Mittelpunkt stehende Class-Datei des Projekts.

Prognose des Aufwands:
Da die Datei im Mittelpunkt steht, wird sie in der Entwicklung dynamisch im Wandel sein. Je nachdem was wir hinzufügen oder entnehmen, werden sich die Methoden und ihr Aufbau ändern.

3.2 Backend

Das Backend besteht, siehe Diagramm am Anfang, aus einer zentralen Soldier-Superclass, die die Objekte im von der Board-Class erstellten 2D-Array sind. Darüber hinaus gibt es eine Coordinates-Class, welche genutzt wird, um positionswerte für das Schachbrett ohne großen Aufwand zu übergeben. Die Damage-Class steht in Verbindung zu Soldier, da die dort definierten Methoden der Schadensvermittlung zwischen Objekten dienen.

3.3.1 Soldier Superclass und Subclasses



Funktion:
Die Superclass Soldier stellt mit seinen Subclasses die verschiedenen Arten der Objekte dar, welche im 2D Array genutzt werden, und anhand derer das Frontend die Texturen des Schachbretts visualisieren kann.
Aufbau:
Die Superclass ist Soldier.class. Sie enthält einen String “teamColor” - welcher im Spiel entweder red oder green ist, egal ob die Teamfarben sich optionsmäßig ändern - und einen integer health, welcher die Lebenszahl darstellt. Soldier enthält get und set methods für die construct-values, sowie eine methode mathMove. Diese kalkuliert anhand einer Position die möglichen Bewegungsfelder im Abstand von einer Kachel. Diese Methoden wird von Objekten dann überschrieben, wenn sich Schachfiguren sich nicht auf diese eine Art bewegt. Die Bewegung mit einer Weite von eins wurde gewählt, da diese die am Meisten vorkommende Bewegungsart ist.

Die Subklassen enthalten, wenn Sie Schachfiguren und keine Füller sind, eine calculateDamage, eine defendAndBleed Methode und eine takeSelfie Methode. Diese werden durch Interfaces mit inbegriffen, da die Soldierclass selbst diese Methoden nicht benötigt.

Calculate Damage kalkuliert zwischen einem min und max Schadenswert Anhand des Math.packages zufällig. Die Methode enthält Positionswerte des Verteidigers und kann so feststellen, ob der Attackiert zusätzlichen Schaden hinzugefügt wird.

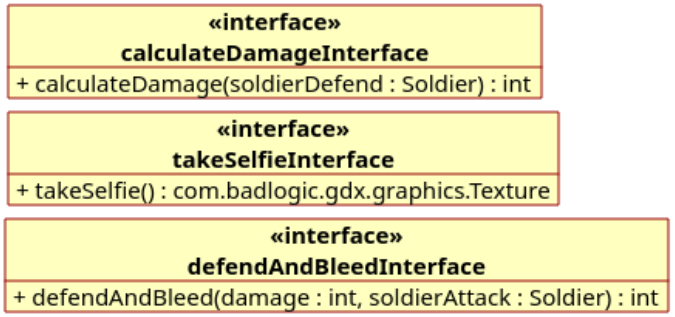
defendAndBleed ist die Methode der Verteidung. Es wird der nativ ermittelte Schadenswert genommen und je nach Angreifer wird der Schaden verringert.

takeSelfie gibt die in BoomChess festgelegten Variablen, in denen die Textur des Aussehens der Subclass steht, zurück.

Position im Program:
Soldier ist ein zentrales Objekt des Backends, welches die Schachfiguren als Objekte darstellt.

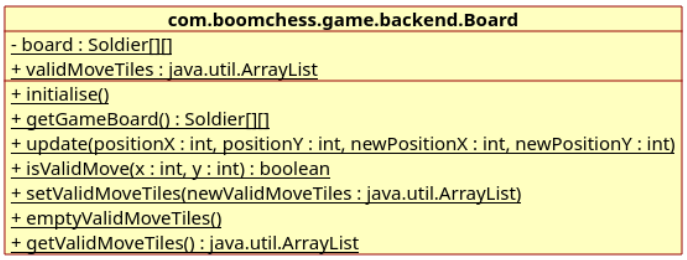
Prognose des Aufwands:
Die Erstellung anhand eines Plans wird vermutlich an die 4 Stunden durch die aufwendigen mathMove Methoden brauchen. Da der Plan nicht spiel-technisch einkalkuliert sein wird, muss mit noch mehr Zeit später in der Entwicklung gerechnet werden, um die Stärke der Schachfiguren auszubalancieren.

3.1.1.1 Interfaces



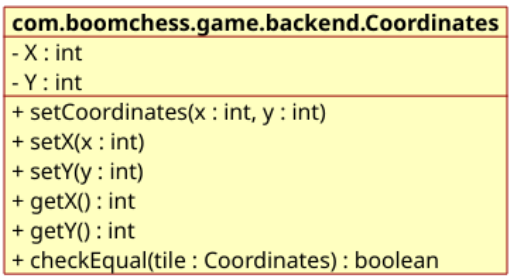
Funktion:
Die Interfaces geben die Funktionalität des bestimmten Methodenaufrufs zu den Subclasses von Soldier, ohne dass die Superclass diese hat (da Sie diese nicht braucht). Dies soll es verhindern, dass wir falsche Werte aus einer Methode bekommen, wenn ein Objekt falsch erstellt/aufgerufen wurde -> Exceptions bemerken und Handeln.
Aufbau:
Die Interfaces enthalten eine simple Anweisung an die Methode.
Position im Program:
Die Interfaces, die oben genannt sind, sind im Backend für die Soldier-Classes der aktiven Spielfiguren gedacht.
Prognose des Aufwands:
Die Erstellung der Interfaces ist straightforward und in wenigen Minuten machbar. Je nach Spielerweiterung können mehr Interfaces mit dem gleichen Einsatzgebiet erstellt werden.

3.2.2 Board



Funktion:
Die Board-Class enthält das Spielbrett als 2D-Array von Soldier.
Aufbau:
initialise(): Erstellt das 2D Array mit Soldier-Subclass-Constructors und einer nicht-parameterabhängigen vorher-festgelegten Struktur des Schachbretts. Siehe Overview am Start des Papers.
update(): tauscht die Position von zwei Soldier Objekten im 2D-Array.
isValidMove(): Nimmt Positionsdaten und schaut, ob diese in ihrer Größe auf die Breite und Höhe des Schachbretts passen.
setValidMoveTiles: nimmt die ArrayList der möglichen Coordinates und setzt Sie als Positionen für das xMarkerOverlay, welches anzeigt, auf welche Positionen sich eine Figur im Drag-Zustand bewegen kann.
Position im Program:
Board befindet sich im Backend und ist eine native Subclass von Object.
Prognose des Aufwands:
Board wird in seiner Erstellung mindestens 2 Stunden brauchen. Darüber hinaus wird sich initialise() dynamisch in der Entwicklung ändern, wenn wir Soldier-Subclasses entnehmen, hinzufügen oder ändern. Zudem kann Sich das Spielkonzept oder die Größe des Schachbretts grundlegend ändern.

3.3.3 Coordinates



Funktion:
Coordinates gibt als Object einen Shortcut für das Halten von Positionsdaten (x, y) für das Schachbrett. Besonders wichtig wird das Object im Berechnen der möglichen Bewegungspositionen, da so keine Tupel verwendet werden müssen und stattdessen eine ArrayList.
Aufbau:
Coordinates besteht aus einem X und Y integer sowie set und get Methods.
Position im Program:
Coordinates ist Teil des Backends und ist eine native Subclass von Object.
Prognose des Aufwands:
Coordinates ist ein kleines Objekt. Zeitaufwand von 30 minuten vorhergesagt.

3.2.4 Special Objects

Die Special Objects sind Spielerweiterung wie der mögliche zufällige Zusatz von Gebirgen, die nicht von Spielcharaktern übergangen werden können, oder die Funktionalität eines einfachen NIKI-Bots über simple Zug-Reihenfolgen bis zu greedy Algorithmen.

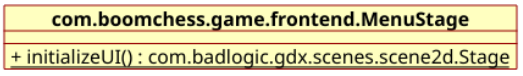
Die Soldier-Class Hill
Hat wie die Empty Soldier-Subclass keine Bewegungs- oder Schaden-soldier-subclass-Methoden, sondern hält eine spezielle Funktion für das 2D-Array. Das Hill-Objekt hält eine TakeSelfie Methode zu einer BoomChess.Java Variable Hill, die eine Texture eines Gebirges enthält. Das Hill Objekt wird bei mathMove Algorithmen nicht als mögliche Spielzug Position registriert und wird als Blockade, an der Kalkulationen enden, behandelt. Sie stellt daher ein nicht valid MoveTile dar
Die Erstellung des Objekts sollte als einfache Hinzufügung eine Stunde dauern.

Die Bot-Class
Die Bot-Class ist ein eigenständiges Java-Objekt, welches Methoden enthält, die zur Berechnung und Coordinate-Object Zugrückgabe genutzt werden,
Ein Zug könnte über mehrere Weisen berechnet werden. Es könnte über alle Objekte des 2D-Arrays gegangen werden, und wenn die Teamfarbe die des aktuellen Zuges entspricht, könnten die möglichen Bewegungen berechnet werden und an jedem möglichen Bewegungspunkt der mögiche zukünftige Schaden ermittelt werden. Nachdem voll in O(n) iteriert wurde, kann der Zug von Objekt zu specific Coordinate mit dem höchsten Schadenswert berechnet werden. Andersweitig könnte nicht nach Schaden, sondern nach einem O(n²) eine zweizügige Prognose abgeben werden, die versucht, so viele Spielfiguren wie nur möglich so nah wie nur möglich an eine Vielzahl an gegnerischen Figuren zu bringen. Als Rückfallebene bei keinem eindeutigen Ergebnis kann ein zufälliges Objekt zufällig ziehen.

3.3 Frontend

3.3.1 Stages

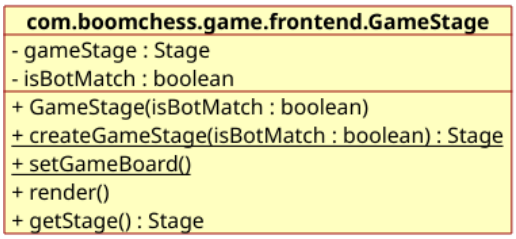
3.3.1.1 menuStage



Funktion:
Das MenuStage ist das erste dargestellte UI-Cluster des Spiels. Es soll Hilfe, Einstellungsoptionen, Spielstart, Credits und Spielbeendigung bieten.
Aufbau:
menuStage besteht aus einer root Tabelle (Package Scene2DUI). In dieser Tabelle bauen wir von oben nach Unten das Menü. Ein Logo, Hilfe-Button, Spielerstellung, Optionen, Credits, Exit.
Diese TextButtons (Scene2DUI) haben sogenannte ActiveLIsteners. Wenn auf Sie geklickt wird, wird das Ereignis-mäßig erkannt und Code ausgeführt. Dieser Code nutzt jeweils die SwitchToStage Methode von BoomChess.Java um eine neue Stage als currentStage festzulegen.
Position im Program:
MenuStage ist die zentrale Stage für die Bedienung des Spiels, Teil des Frontends und ist eine subClass von Stage, einem scene2DUI Package Object.
Prognose des Aufwands:

MenuStage wird mit GameStage eine der aufwendigsten Stages. Die Erstellung kann mit Auswahl des Skins (wie die Buttons aussehen, festgelegt in JSON und PNGS) bis zu fine-tuning der TextButtons bis zu 3 Stunden dauern. Zudem wird das Erstellen des BoomLogos zeitaufwendig werden, falls es melodisch mit dem Hintergrund übergehen soll.

3.3.1.2 gameStage

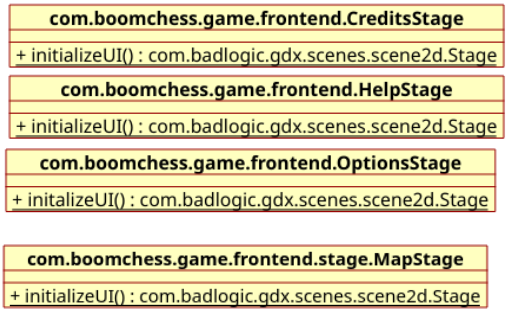


Funktion:
Die GameStage ist die für das Spiel genutzte UI-Stage. Mit ihr wird das Schachbrett dargestellt. Sie soll die Funktionalität von Drag&Drop und einem Mögliche-Schritte-Overlay enthalten.
Aufbau:
Die GameStage wird kreirt, indem eine Tabelle zentriert auf dem Bildschirm aufgebaut wird und in ihrem Aufbau das 2D-Array spiegelt. Die Methode createGameStage iteriert über das GameBoard (2D-Array) und für jedes Objekt wird ein 80 x 80 pixel große Kachel mit einer Textur und ActiveListeners für Drag und Drop erstellt. Dem Objekt wird über takeSelfie seine festgelegte Textur entzogen und als Image dargestellt. Diese kann bei erneutem Aufruf von createGameStage automatisch garbaged werden. Nach jeder vollständigen Iteration über die Zeile von gameBoard, wird auch eine neue Zeile in der Tabelle erstellt. Am Ende spiegelt die UI das gameBoard wieder.
Wenn es eine Aktualisierung gab, wie:

- Falsches Team versucht zu bewegen (if clauses in Drag and Drop)
- Es ein Positionsupdate gab (Board.update)
- Es ein Damage.killPiece() gab.

Soll createGameStage erneut mit BoomStage.SwitchToStage() aufgerufen werden, um das gesamte Overlay mit der Realität von gameBoard neu zu laden.
Neben ActiveListeners soll jede Kachel auch einen Lebensbalken haben, wenn die Maus über der Figur ist. Der benötigte Wert wird über Soldier.getHealth() bei der Iteration beim jeweiligen Objekt erhalten.
Position im Program:
gameStage befindet sich im Frontend und ist eine subClass von Stage, einem scene2DUI Package Object.
Prognose des Aufwands:
gameStage wird mit Abstand am Längsten von den Stages benötigen. Eine mögliche Prognose ist 10 Stunden. Der Algorithmus der Erstellung des Schachbretts, der Listeners, des Drag&Drops, des Overlays für mögliche Bewegungen wird komplex.

3.3.1.3 restlich Stages



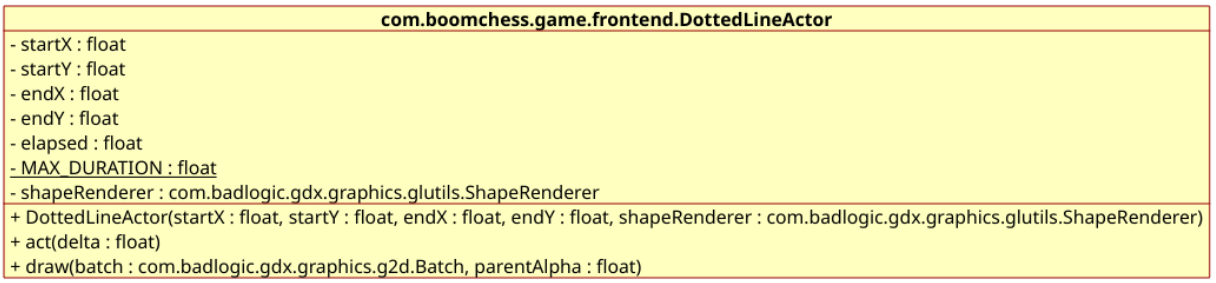
Funktion:
Die CreditsStage, HelpStage und OptionsStage werden den Namen entsprechend aus dem menü aufgerufen werden und ihre Funktionalität als Stage erfüllen, bevor sie mit einem Back Button zurück das mainMenu erstellen.
Aufbau:
Creditsstage soll eine große Textbox mit allen teilhabenden Personen und Quellen/Tools für Erstellung der Bilder, Sounds etc enthalten.
HelpStage soll ein großes PNG mit allen wichtigen Spielinfos zu Optionen und Schachfiguren enthalten.
Die MapStage soll je nach Größe des Bildschirms das Schachbrett unterhalb der Spielfiguren laden und sich verändern können, wenn ein neues RandomImage als map geladen wird.
Die OptionsStage soll es ermöglichen, Spielmodi, Schachfiguren-Farben und Hindernisse einzustellen. Wenn bestimmte TextButton gedrückt werden, sollen Spielvariablen von BoomChess geändert werden - wie die in Variablen hinterlegten Texturen oder ein boolean zu “areObstacles”.
Position im Program:
Prognose des Aufwands:
Die restlichen Stages werden mit Ausnahme von OptionStage schnell machbar sein. OptionStage wird in Verbindung der Erstellung zusätzlicher Texturen länger dauern.
Insgesamt 3 Stunden prognostiziert.

3.3.1.3.1 Besonderheit audioStage

Funktion:
Die audioStage ist ein besondere Stage, die jederzeit in der linken unteren Bildschirm-Ecke gerendered werden soll. Sie soll die Möglichkeit bieten, die Musik auszuschalten, alle Audio-Inhalte stummzuschalten und über Sliders die Lautstärke von Musik und Sounds anzupassen.
Aufbau:
Der Aufbau ist simpel: Eine Tabelle, die in der oberen Reihe ein Button für Musik aus und Sound aus hat und in den unteren Reihen Sliders für die Lautstärkeregulierung. Alle hier genannten UI-Elemente sind Teil von Scene2DUI und können über die skin variable designed werden.
Position im Program:
Die AudioTable ist Teil von BoomChess.java, Teil des Frontend, und nicht seine eigene Class.
Prognose des Aufwands:
DIe AudioTable wird in 1-2 Stunden programmiert und implementiert sein.

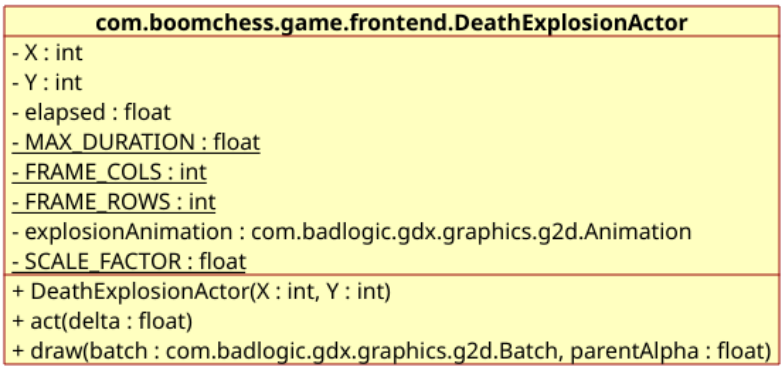
3.3.1 Actors

3.3.1.1 dottedLine



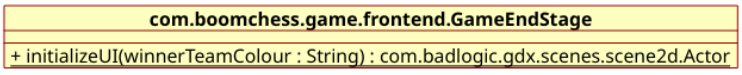
Funktion:
Der DottedLineActor ist ein Actor-Objekt, das jedes mal aufgerufen wird, wenn zwischen zwei Soldiers ein Schaden entsteht. Der DottedLineActor soll ein Shape (geometrischen Object) renderen/zeichnen, zwischen den zwei im Schaden involvierten Soldiern. Der angreifende Soldier soll einen kleinen blauen Punkt als Attackierer erhalten.
Aufbau:
DIe Start floats sind die Positionsdaten des Attackieres, die end floats die des Verteidigers. Sie sollen als Pixelkoordinatenwerte übergeben werden. Um diese zu erhalten, werden die Methoden calculatePixelbyTile in BoomChess vor Parameterübergabe genutzt.
Elapsed ist eine Variable, welche durch deltaGetTime jede vergehene echtzeitliche Zeiteinheit erhöht wird.
MAX_DURATION ist ein festgelegter maximal Wert, wie groß Elapsed sein darf, bis das Objekt nicht mehr auf dem Bildschirm angezeigt werden soll.
Der ShapeRenderer ist ein aus dem GLUTILS Package genutztes Objekt, welches es ermöglicht, Objekte mit Orientierung und Form zu erstellen. Die dottedLine wird, wie der Name vermuten lässt, aus festgelegten Rechtecken mit festgelegten Abstand über eine for loop zwischen den Pixelkoordinaten gezeichnet.
Position im Program:
Der DottedLineActor ist eine Actor subclass und Teil des Frontends. Sie wird von BoomChess & Damage aufgerufen, falls Schaden zwischen zwei positionen passiert.
Prognose des Aufwands:
Der DottedLineActor ist komplexer und grundlegend Anders als zuvor erstellte Objekte der UI. Eine Erstellung kann mindestens 2 Stunden, ohne Einberechnung der calculatePixelbyTile Methoden dauern.

3.3.1.2 Animation-Actors, am Beispiel DeathExplosion



Funktion:
Der DeathExplosionActor ist eine Animation, die an einem bestimmten PX Punkt stattfindet, wenn eine Schachfiur kein Leben mehr hat.
Aufbau:
Als Actor Object, welches eine Animation enthält, muss das Objekt wie der DottedLineActor eine elapsed Zeit Variable und eine MAX_DURATION enthalten. Für die Animation muss passend zum sogenannten SpriteSheet (ein PNG mit allen Frames der Animation) initalisiert und passend wie eine Tabelle die Reihen und Columns festgelegt werden. Die Animation selbst ist dabei eine Subclass des Objekts GDX.Animation, was eine Nutzung von Spritesheets ermöglicht.
Der sogenannte SCALE_FACTOR bezeichnet einen float value, der mit der Größe der Animation multipliziert wird, um diesen in seiner Size zu ändern.
Die Position X und Y wird durch calculatePixelByTileNonGDX erhalten, da die PX Koordinate für die Animationsplatzierung von unten Links des Bildschirms und nicht von oben links ausgeht.
Position im Program:
Der DeathExplosionActor ist Teil des Frontends und wird mit addToStage dem currentStage für die dauern von MAX_DURATION hinzugefügt.
Prognose des Aufwands:
Mit der Erstellung des Spritesheets und der Initialisierung der passeden Werte nach Erstellung des eigentlichen Codes, wird eine Länge von 3 Stunden prognostiziert.

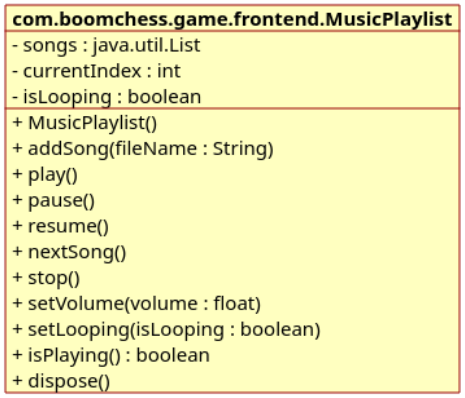
3.3.1.3 GameEndStage (Actually an Actor for Stage-adding)



Funktion und Aufbau:
Der gameEndStage wird als Actor dem currentStage hinzugefügt. Er wird getriggert, wenn ein general weniger oder gleich 0 Leben hat. Er erhält einem dem Bildschirm überlagerten Text, in dem das Gewinnerteam bejubelt wird und es eine zusätzliche TextButton Möglichkeit geben soll, zurück zum Menü zu gehen.
Position im Program:
Die GameEndStage ist als Object Actor subclass ein Teil des Frontends. Es wird am Ende der Spielrunde aufgerufen.
Prognose des Aufwands:
Die Erstellung unterliegt keiner Komplexität und sollte max. 1 Stunde dauern.

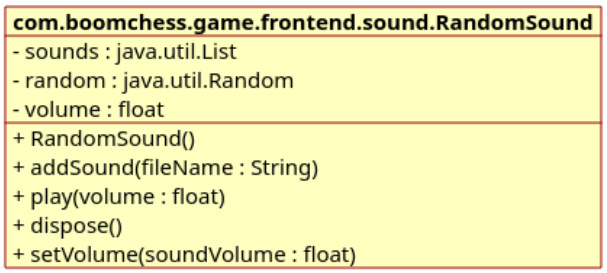
3.3.3 Other Objects

3.3.3.1 MusicPlaylist



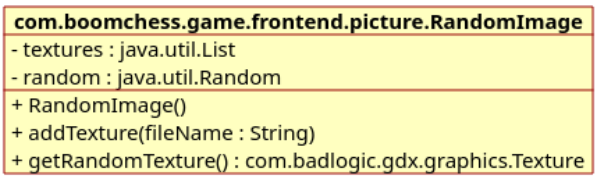
Funktion: MusicPlaylist soll die Funktionalität erhalten, die am Anfang in BoomChess.Java geladenen Musikfile zu halten und Methoden wie “Play”, “Pause”, “Resume”, “Next”, setVolume etc. zu besitzen.
Aufbau: MusicPlaylist hat hierhingehend zum Beispiel den Aufbau, dass seine Variablen “Songs”: Eine Liste von Songs enthält, die mit addSong hinzugefügt werden können, “currentIndex”: der derzeit abzuspielende Song und “isLooping”: boolean Zustand ob in einem Loop spielbar. Die in der List befindlichen Objekte sind gdx “Music”-Objects. Bei Auswahl des nächsten Songs soll zufällig über das java Random Package ein zufälliger Index ausgewählt werden.
Position im Program: Im Frontend befindliches Objekt, interagiert mit BoomChess.java
Prognose des Aufwands: Die Erstellung des Codes 1-2 Stunden, die Musikauswahl kann deutlich länger sein

3.3.3.2 RandomSound



Funktion/Aufbau:
RandomSound ist ein Objekt des Frontends welches gdx.audio Sound Objekts zufällig aus eine Liste gibt, welches diese in BoomChess über die RandomSound.addSound method erhalten hat. Die Zufälligkeit wird durch ein Nutzen des java.util Random Package ermöglicht.
Aufbau:
Position im Program:
Das Objekt ist Teil des frontends und wird in BoomChess.Java als public static Variable erstellt, in create initialisiert und dort auch mit Sound-Objects gefüllt. In den spezifischen Soldier-Subclassen werden die passenden RandomSound.Objects dann zum Abspielen der Soundeffekte genutzt.
Prognose:
Das Programmieren dieses Objektes kann bis zu 2 Stunden + 1 Stunde debuggen dauern, da das Nutzen von GDX-Methods passend zum Sound-Objekt komplex und nicht straightforward werden kann.

3.3.3.3 RandomImage



Funktion:
RandomImage ist ein Objekt, dass in einer Liste zufälig zuvor hinzugefügte Texturen auf Befehl ausgeben soll, diese aber nur über einen Index ausgeben und nicht nach Nutzung löschen.
Genutzt soll dieses Objekt in den Schachbrettkarten des Spiels, die über einen Knopfdruck änderbar sein sollen.
Aufbau:
RandomImage ist der kleine unkomplizierte Bruder von RandomSound. Über addTexture wird über den Speicherort des pngs dieses als Texture Objekt in eine Liste geladen. Über getRandomTexture wird, wenn etwas in der Liste ist, ein Objekt ausgegeben. Um Exceptions entgegenzuhandeln, wird eine IllegalStateException geworfen, wenn keine Texturen verfügbar sind.
Position im Program:
RandomImage ist Teil des Frontends.
Prognose:
Die Erstellung des RandomImage Objekts sollte eine Stunde dauern. Die Texturen-Objekte können unkompliziert und ohne Behandlung übergeben werden, da Sie nach Aufruf zu wegwerfbaren Images transformiert werden.