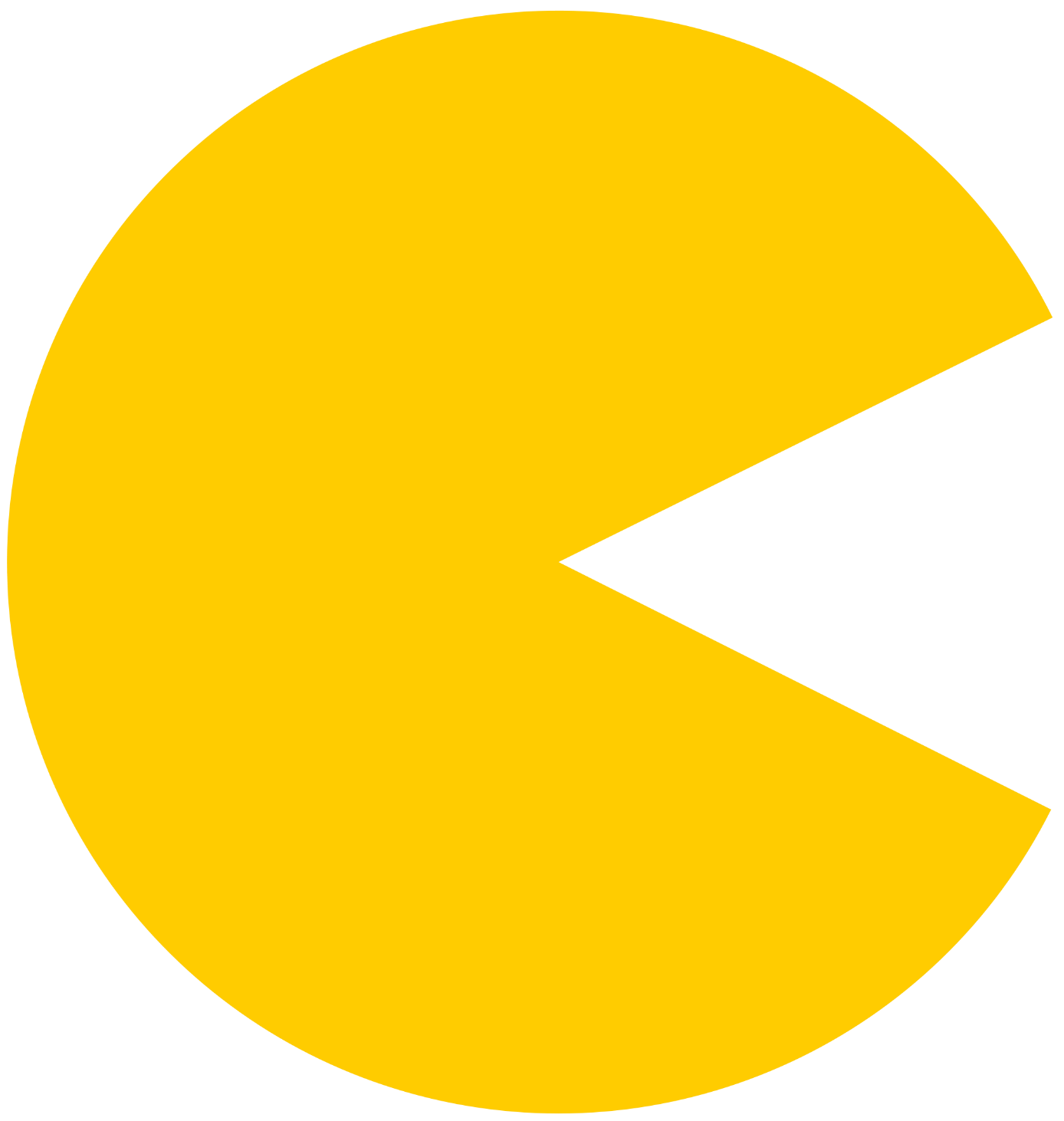
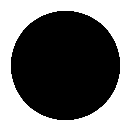
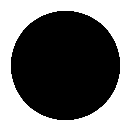
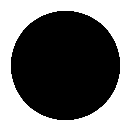
**Pacman 2.0 – Dokumentation**

**Tobias Scholz**









**Gliederung**

1. Zielbestimmung … 3
2. Produkt-Einsatz … 3
3. Produkt-Umgebung … 4
4. Produkt-Funktionen … 4
5. Produkt-Daten … 4
6. Produkt-Leistungen … 9
7. Benutzungsoberfläche … 9
8. Qualitätsziele … 12
9. Testfälle … 14
10. Entwicklungsumgebung … 15
11. Ergänzungen … 15
12. Scrumboard … 16

**1. Zielbestimmung**

Muss-Kriterien:

Es sollte ein funktionierender Pacman-Nachfolger erstellt werden, der sich nah am Original orientiert, aber trotzdem noch eigene Features enthält. Es muss also ein Level existieren, in dem sich der Spieler frei und nachvollziehbar bewegen kann, aber auch die Gegner. Für die Gegner sollte eine relativ intelligente, aber trotzdem noch faire KI erstellt werden, damit diese innerhalb des Levels den Spieler verfolgen kann. Für die praktische Handhabung des Nutzers muss auch eine Menüstruktur vorhanden sein, mit welcher auf das Scoreboard zugegriffen werden kann und das Spiel gestartet und auch neugestartet werden kann. Auf dem Spielfeld selber sollen Münzen als Collectables liegen, aber auch Power-Ups, die einem bei der Flucht vor den Gegnern helfen.

Wunsch-Kriterien:

Das Spiel soll einen qualitativ hochwertigen Eindruck hinterlassen. Deswegen ist das Erstellen eines ansehnlichen Designs ein weiteres Ziel. Alle Teile der Benutzer-Oberfläche sollen einem einheitlichen Design-Stil folgen.

Zusätzlich soll es möglich sein, das Spiel jederzeit auf Knopfdruck zu pausieren. Ein Punktezähler an der Seite zeigt zu jedem Zeitpunkt den Punktestand.

Ausschluss-Kriterien:

Es hätte noch verschiedene Typen von Gegnern geben können mit verschiedenen Eigenschaften und Aussehen. Außerdem könnte man für die Zukunft noch viele weitere Level designen mit eventuell höheren Schwierigkeitsgraden.

Auf Grund fehlender Zeit ist leider auch das grafische Drehen des „Pacmans“, wenn dieser im Spiel seine Richtung ändert, ausgelassen worden.

**2. Produkt-Einsatz**

Anwendungsbereich:

Das Spiel hat keinen konkreten Anwendungsbereich. Jeder kann das Spiel auf jedem beliebigen PC installieren und nutzen, um Langeweile zu bekämpfen.

Nutzergruppen:

Das Spiel ist für alle Nutzergruppen fast jeden Alters gedacht. Es wird keine spezielle Qualifikation für die Nutzung benötigt.

**3. Produkt-Umgebung**

Software:

Das Programm läuft auf allen Windows7 und Windows10 Systemen. Zum Starten benötigt das Spiel alle Texturen im „bin“-Ordner. Alle Dateien zur Speicherung der Scoreboard-Daten erstellt das Programm eigenständig.

Hardware:

Zum Starten sollte eine dedizierte Grafikkarte oder mindestens eine moderne OnBoard-Grafikkarte im Rechner eingebaut sein. Zur Bedienung benötigt man eine Maus und eine Tastatur.

Schnittstellen:

Es sind keine Schnittstellen zu anderen Systemen vorhanden.

**4. Produkt-Funktionen**

Funktion 1: Auswahl der Schwierigkeitsstufe mit dem „Schwierigkeitsstufen“-Button

Funktion 2: Starten/Pausieren/Entpausieren/Neustarten mit dem „Start…“-Button

Funktion 3: Aufrufen des Scoreboards bei Betätigung des „Scoreboard“-Buttons (laden der Daten aus der „Scoreboard.txt“-Datei)

Funktion 4: Initialisierung des Levels (Platzieren von PowerUps und Münzen, Herausfinden des Spieler- und Gegner-Spawns, Festlegung der Farben der Kollisions-Map)

Funktion 5: Steuern des „Pacmans“ mit den WASD-Tasten, aufsammeln von Münzen und PowerUps, Zerstören der Gegner (wenn diese angreifbar sind)

Funktion 6: Speicherung eines möglichen Highscores in der „Scoreboard.txt“-Datei

Funktion 7: Steuerung der Gegner mit einer KI, die ein Verständnis von ihrer Umgebung und der Position des Spielers hat

**5. Produkt-Daten**

Daten, die aus externen Dateien geladen werden:

D1/ - name, Datentyp-String, alle Zeichen möglich außer ein leerer String

D2/ - topPunktzahl, Datentyp-int, 0 – 4294967295

D3-26/ - Datentyp-Image, Texturen für alle Sprites,

Andere wichtige Daten:

D27/ powerUps, Datentyp-ArrayList<SpeedPowerUp>, Speicherung aller PowerUps

D28/ coins, Datentyp-ArrayList<Coin>, Speicherung aller Münzen

D29/ geister, Datentyp-ArrayList<Geist>, Speicherung aller Geister

D30/ player, Datentyp-Player, Speicherung des Spielers

D31/ richtung, Datentyp-int, 0-3, Speicherung der Richtung eines bewegenden Gegenstands

D32/ momentaneTextur, Datentyp-int, die momentane Textur des Pacmans

D33/ - position, Datentyp-Vec2d, Speicherung der Position der Gegenstände

D34/ - level, Datentyp-Level, Speicherung der momentanen Level Instanz

D35/ - spielerSpawnVec, Datentyp-Vec2d, Position des Spieler Spawns im Level

D36/ - gegnerSpawnVec, Datentyp-Vec2d, Postion des Gegner Spawns im Level

D37-40/ - wand/spielerSpawn/gegnerSpawn/gegnerOnlyZugang, Datentyp-Color, Festlegung der Farbe auf der KollisionsMap

D41-42/ - width/height, Datentyp-int, 0 – 4294967295, Größe des Levels in Pixeln

D43/ - status, Datentyp-int, 0-3, Festlegung des momentanen Spiel-Status

D44/ - playerTPS, Datentyp-int, 60/90, Anzahl der Player-Updates pro Sekunde

D45/ - geisterAngreifbar, Datentyp-boolean, legt fest, ob die Gegner angreifbar sind

D46/ - speedAktiviert, Datentyp-boolean, legt fest, ob gerade ein PowerUp aktiviert ist

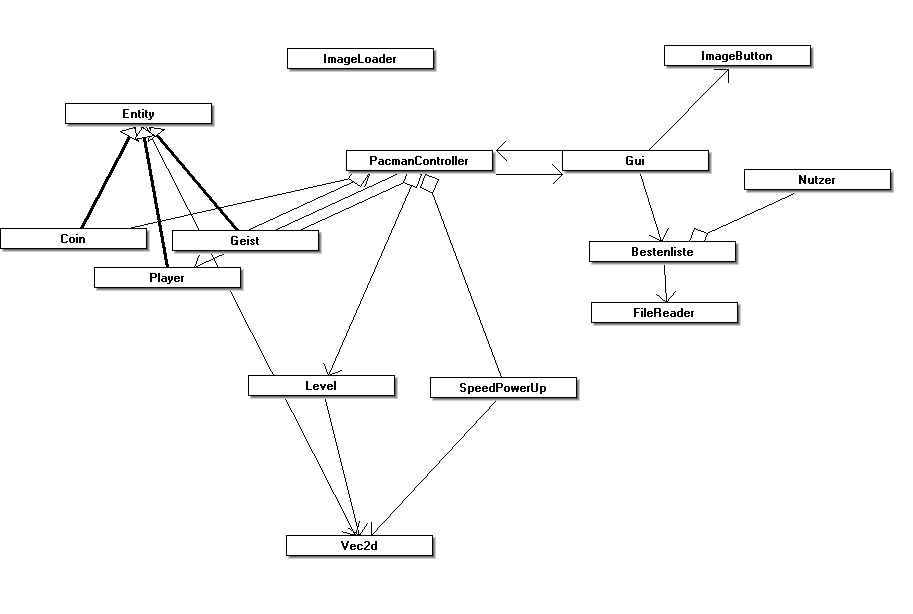
D47/ - requestedDirection, Datentyp-int, 0-3, die vom Nutzer angefragte nächste Richtung für den Pacman

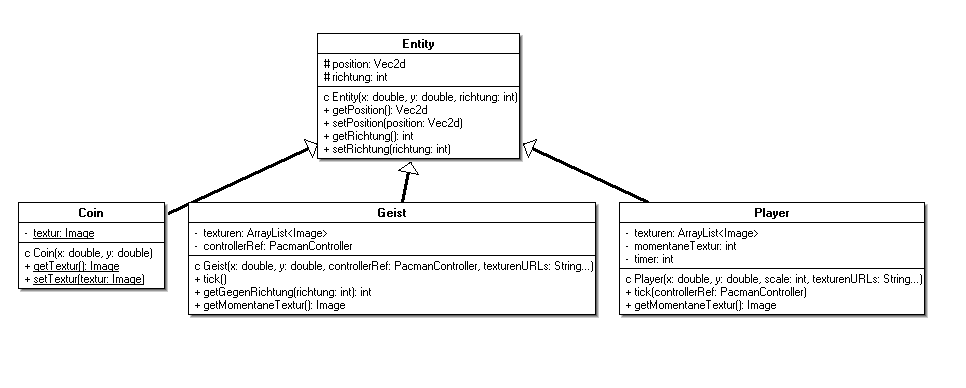
D48/ - guiRefreshed, Datentyp-boolean, wenn true, dann soll die Gui refresht werden

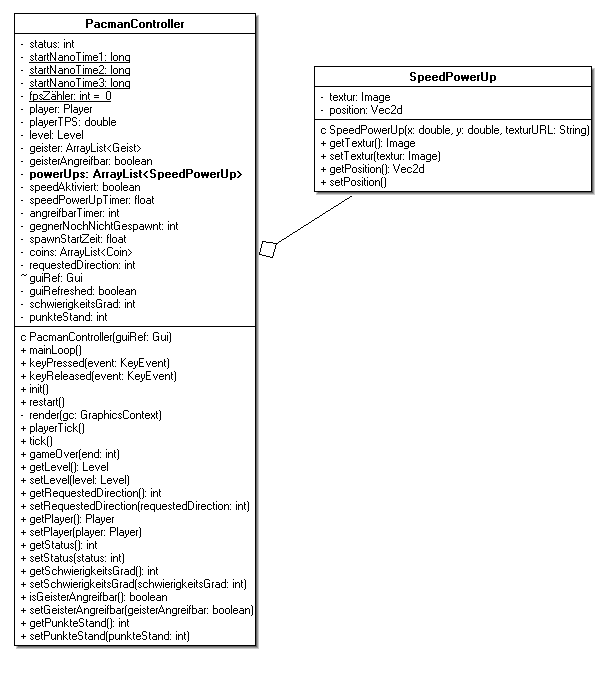
D49/ - schwierigkeitsGrad, Datentyp-int, 0-2, legt den Schwierigkeitsgrad des Spiels fest

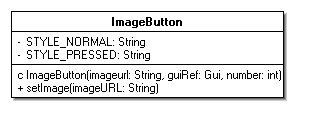
D50/- punkteStand, Datentyp-int, 0 – 4294967295, Wert des momentanen Punktestands

UML-Klassendiagramm

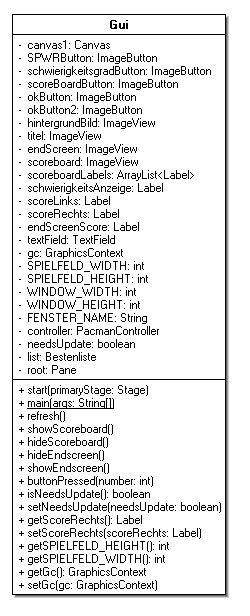
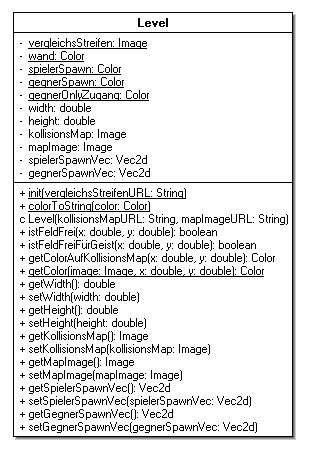
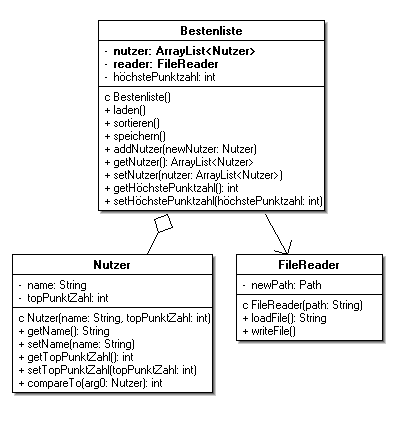












**6. Produkt-Leistungen**

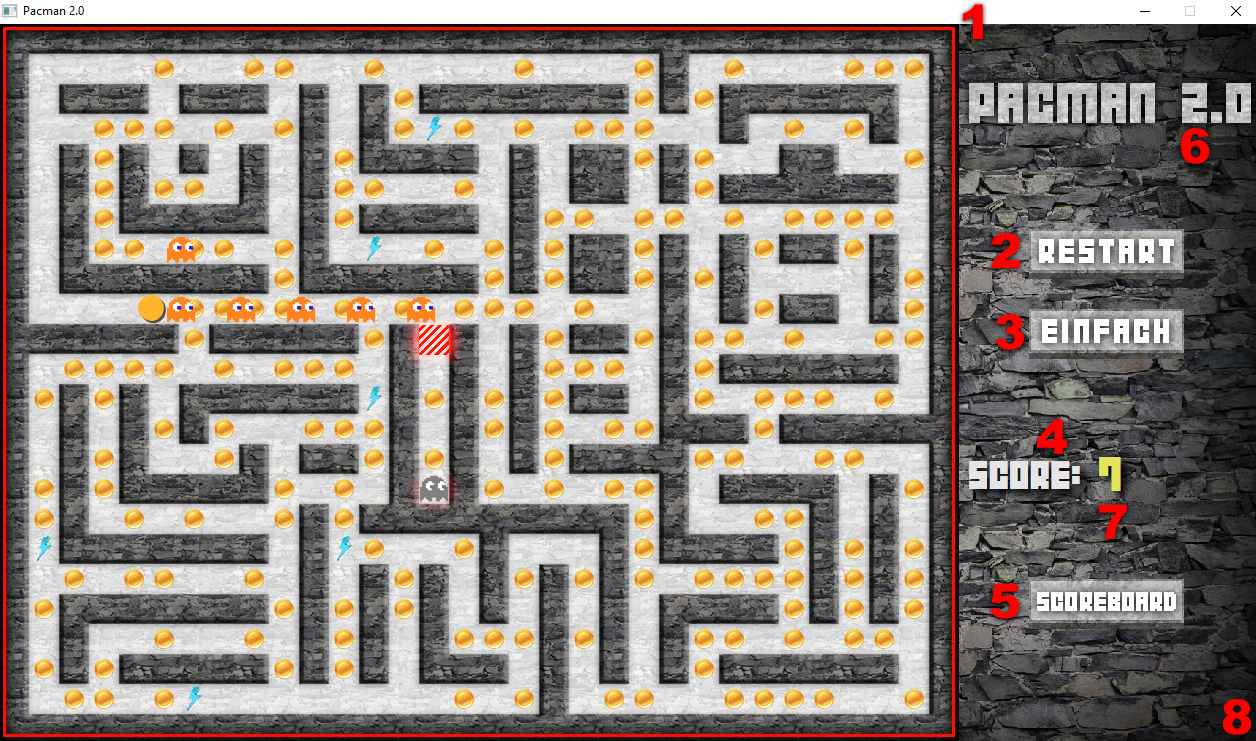
Wichtig bei dem Spiel ist die Performance. Bei diesem Spiel ist die Performance im normalen Bereich, wenn zu jedem Zeitpunkt die Main-Schleife 90 mal pro Sekunde durchlaufen wird. Sollte diese Zahl unterschritten werden, verlangsamt zunächst erstmal die Logik des Spiels. Dies würde sich dadurch bemerkbar machen, dass sich die Pacman-Spielfigur langsamer bewegen würde als sonst beim Aufsammeln eines PowerUps. Wird die Schleife weniger als 60 mal pro Sekunde durchlaufen werden, verlangsamt sich das gesamte Spiel und es werden auch weniger Bilder pro Sekunde dargestellt. Jedoch sollte ein durchschnittlich leistungsfähiger Computer die 90 Updates pro Sekunde leisten können.

Bei dem Testen des Spiels auf meinem Rechner sind also keine merkenswerten Verlangsamungen aufgefallen. Bei Betätigung jeglicher Knöpfe erfolgen die Aktionen ohne Verzögerung.

Eine Runde „Pacman 2.0“ kann wenige Sekunden dauern, kann aber auch mehrere Minuten gehen. Man spielt, solange einem nicht langweilig wird.

7. Benutzeroberfläche

Bildschirmlayout:



|  |  |  |
| --- | --- | --- |
| Nr | Komponente | Beschreibung |
| 1 | **Canvas:**  canvas1 | Grafische Darstellung aller Objekte des Spiels mit Hilfe der Klasse „Graphicscontext“ |
| 2 | **Button:**  SPWRButton | Startet/Pausiert/Entpausiert/Restartet das Spiel, Funktion und Bild des Buttons ändern sich je nach momentanem Status des Spiels |
| 3 | **Button:** schwierigkeitsgradButton | Toggled den Schwierigkeitsgrad des Spiels, Bei Knopfdruck wird zum nächst höheren Grad gewechselt (beim höchsten dann zum niedrigsten) |
| 4 | **Label:**  scoreLinks | Stellt den Text „Score: “ dar |
| 5 | **Button:**  scoreBoardButton | Bei Betätigung öffnet sich rechts auf dem Bildschirm das Scoreboard |
| 6 | **Label:**  titel | Stellt den Text „Pacman 2.0“ dar |
| 7 | **Label:**  scoreRechts | Holt sich aus der Controller-Klasse die Information über den momentanen Punktestand und stellt sie in gelber Farbe dar |
| 8 | **ImageView:** hintergrundBild | Hintergrundbild für das gesamte Fenster |



|  |  |  |
| --- | --- | --- |
| Nr | Komponente | Beschreibung |
| 9 | **Button:**  okButton | Schließt das „GameOver“-Fenster, gegebenenfalls sorgt er für die Speicherung eines neuen Highscores |
| 10 | **TextField:**  textField | Sollte der Nutzer den vorherigen Highscore überboten haben, kann er hier seinen Namen eingeben, damit er im Scoreboard gespeichert ist |
| 11 | **ImageView:**  endScreen | Hintergrund-Bild für „GameOver“-Fenster |
| 12 | **Label:**  endScreenScore | Fragt den Punktestand aus der Controller-Klasse ab und stellt ihn dar |



|  |  |  |
| --- | --- | --- |
| Nr | Komponente | Beschreibung |
| 13 | **ArrayList<Label>:**  scoreboardLabels | Darstellung der Daten aus der „bestenliste.txt“ |
| 14 | **Button:**  okButton2 | Schließt bei Betätigung das Scoreboard-Fenster |
| 15 | **ImageView:**  Scoreboard | Hintergrundbild für das Scoreboard |

MVC-Konzept:

Um die View kümmert sich in dem Programm die „Gui“-Klasse. Diese stellt alle Daten des Programms in grafischer Form dar. Vor allem gehört bei diesem Programm dazu ein „Canvas“, der alle Gegenstände des Spiels mit Hilfe der Klasse „GraphicsContext“ mit Texturen darstellt. Die Klasse informiert ebenfalls die Controller-Klasse über alle nötigen Nutzereingaben. Alle notwendige Information fragt die View von dem Model ab. Das Model sind alle Daten des Programms, die sich mithilfe eines UML-Klassendiagramms darstellen lassen. Der Controller ist diesmal nicht mit in der View-Klasse mit eingebaut, sondern hat seine eigene Klasse „PacmanController“. In dieser Klasse befindet sich die gesamte Logik des Spiels, wo alle Abläufe gesteuert werden.

Tastaturbelegung:

W - bewegt die Spielfigur nach oben, solange oben frei ist

A - bewegt die Spielfigur nach links, solange links frei ist

S - bewegt die Spielfigur nach unten, solange unten frei ist

D - bewegt die Spielfigur nach rechts, solange rechts frei ist

Die Tasten müssen nicht gedrückt gehalten werden. Die Spielfigur bewegt sich so lange von alleine, bis eine Wand im Weg ist. Es wird abgeraten, mehrere Tasten gleichzeitig zu drücken, da die Figur sich nur in maximal eine Richtung gleichzeitig bewegen kann.

Mit der Maus werden alle Menüelemente bedient.

Dialogstruktur:

Es gibt 2 verschiedene mögliche Dialoge:

1. Bei Spielende wird das „GameOver“-Fenster geöffnet. Der Nutzer muss auf den „OK“-Button drücken um fortzufahren. Danach ist das Spiel in einem Zustand, in dem es wieder vollständig bedient werden kann.
2. Beim Betätigen des Scoreboard-Buttons wird das Scoreboard-Fenster geöffnet. Durch das Betätigen des „OK“-Buttons kann dieses wieder geschlossen werden.

**8. Qualitätsziele**

Das System entspricht den Mindest-Anforderungen, die ein Nutzer an ein Spiel haben kann. Es hat eine eindeutige Struktur mit fest definierten Zielen. Die Steuerung funktioniert, wie man es sich vorgestellt hat, und auch die Menuführung ist verständlich.

Zuverlässigkeit:

Bis auf wenige Ausnahmen läuft das Spiel einwandfrei. Selten kommt es direkt nach Spielstart zu einer Exception im JavaFX-Application Thread. Die Ursache dieser Exceptions liegt wahrscheinlich an der Schwierigkeit des MultiThreadings in Verbindung mit JavaFX-Applikationen. Zum Glück treten diese nur sehr selten auf.

Integrität:

Das Produkt ist vollständig vor Missbrauch geschützt. Selbst die gespeicherte Bestenliste lässt sich von außen nur schwer manipulieren, da die Inhalte verschlüsselt sind.

Benutzerfreundlichkeit:

Die Anwendung ist sehr benutzerfreundlich. Es gibt zum Beispiel keine Ladezeiten. Jede Aktion führt direkt zum gewollten Ergebnis. Der Benutzter kann direkt das Spiel starten, ohne aufgehalten zu werden. Außerdem ist die gesamte GUI sehr übersichtlich und ansehnlich. Der Benutzer weiß zu jedem Zeitpunkt genau, was er tun kann, und was nicht.

Robustheit:

Bei dem Programm sind keine Falscheingaben möglich, da der Input des Spielers sehr geringgehalten ist. Zu jedem Zeitpunkt sind nur die Knöpfe sichtbar, die auch gedrückt werden sollen. Bei dem TextField sind alle Eingaben erlaubt. Gibt der Nutzer nichts ein, wird keine Speicherung vorgenommen.

Effizienz:

Das Spiel benötigt relativ viele Systemressourcen während der Laufzeit, was die CPU angeht. Im RAM belegt es nur ~141MB und zum Speichern der gesamten Dateien werden nur ~15MB benötigt.

Revisionsfähigkeit:

Änderbarkeit / Erweiterbarkeit:

Was die Revisionsfähigkeit angeht, ist das Spiel sehr gut änderbar, erweiterbar und ausbaufähig. Durch die Strukturierung aller „Dinge“ in verschiedene Klassen und Oberklassen können jederzeit weitere Klassen erstellt werden und in wenigen Schritten eingebunden werden.

Wartbarkeit:

Der Quellcode kann jederzeit einfach gewartet werden. Das Programm selbst gibt hilfreiche Fehlerinformationen aus, z.B wenn eine bestimmte Datei nicht geöffnet oder gefunden werden kann.

Überprüfbarkeit:

Das Spiel lässt sich relativ einfach überprüfen. Nach dem Start können alle groben Funktionen innerhalb weniger Sekunden getestet werden. Da aber sehr viele Szenarien in einem Spiel solcher Komplexität eintreten können, kann man nicht direkt alle möglichen Abläufe testen. Manche Fehler erkennt man erst nach längerer Spielzeit, oder wenn ein Tester das Spiel auf seine eigene Weise spielt und Fehler findet, die man selbst nie entdeckt hätte.

Transitionsfähigkeit:

Portabilität:

Das Programm könnte mit wenig Aufwand auch auf jedem anderem aktuellen Java-Fähigen Gerät gestartet werden. Zur Nutzung bräuchten die Geräte nur Maus, Tastatur, Bildschirm sowie die Installation von Java 8 und man sollte darauf achten, dass die Texturen im richtigen Ordner bereitliegen.

Wiederverwendbarkeit:

Aufgrund der Spezifität des Projektes kann man eher wenige Klassen in anderen Projekten wiederverwenden. Zu diesen Klassen gehören z.B. die „Utility“-Klassen „Vec2d“, „ImageLoader“ und „FileReader“, die auch problemlos in jedem anderen Projekt ihre Aufgabe erfüllen würden.

Interoperabilität:

Das Produkt kann im momentanen Zustand keine Daten mit anderen System austauschen oder mit anderen zusammenarbeiten, da es ein eigenständiges Programm ist.

**9. Black-Box-Test**

Zum Testen des Spiels habe ich mich für den Black-Box-Test entschieden. Dabei habe ich die verschiedenen Funktionen der Menüstruktur getestet, das Spiel selbst und auch die Speicherung von Daten im Scoreboard. Bei dem Testen des Menüs und des Spieles an sich lassen sich keine Tests mit konkreten Werten durchführen, da es nicht wirklich um die Verarbeitung von Daten, sondern mehr um die Verarbeitung von Eingaben von Peripheriegeräten geht.

Test der Menüstruktur:

**Scoreboard-Button**: Lässt sich zu jedem Zustand des Programms drücken und verdeckt/deckt das Scoreboard auf.

**Schwierigkeits-Button**: Lässt sich nur drücken, wenn das Spiel nicht läuft. Also kann das Spielgeschehen nicht durch ein nachträgliches Drücken des Buttons manipuliert werden. Wenn das Spiel gestartet wurde, wird er durch einen Schriftzug ersetzt.

**Start-Button**: Ändert bei jedem Zustand des Spiels seine Funktion und Aussehen. Kann jederzeit problemlos gedrückt werden und erfüllt seine Funktion.

**2 Ok-Buttons**: Nur sichtbar, wenn die jeweiligen Teile der GUI sichtbar sind. Sie schließen beide ihr jeweiliges Fenster und sind nur dann sichtbar, wenn sie auch sichtbar sein sollen.

Test des Spiels:

Zum Testen habe ich externe Personen gefragt, das Spiel zu testen, da diese auch Fehler finden könnten, an die man selbst nie gedacht hätte.

Beim Testen fielen noch Fehler bei der Bewegung des „Pacmans“ auf. Beim Drücken bestimmter Tastenkombinationen konnte dieser durch Wände gehen, da keine Kollisionsabfrage stattfand. Nach kurzer Fehlersuche war dieser Fehler aber auch schnell behoben.

Nachdem mehrere Tester das Spiel einmal durchgespielt haben und keine weiteren Fehler auffielen, gehe ich davon aus, dass keine, oder nur noch sehr gut versteckte Fehler im Spiel vorhanden sind.

Test der Dateneingabe für das Scoreboard:

Der Blick auf das Scoreboard im Spiel verrät, dass es sich um zwei verschiedene Daten handelt. Einmal für einen Namen und für den höchsten erreichten Punktestand dieser Person.

Die Punktzahl wird durch das Programm ermittelt und spiegelte im Scoreboard auch immer die Zahl wieder, die im Spiel erspielt wurde.

Die Angabe des Namens ist allerdings abhängig vom Nutzer. Bestmöglichst sollte das Programm nur richtige Namen erkennen und Eingaben mit Sonderzeichen oder ähnlichem nicht akzeptieren.

|  |  |  |
| --- | --- | --- |
| Eingabe | Erwartetes Ergebnis | Ergebnis |
| „Hans“ | Speicherung | Speicherung |
| „Peter“ | Speicherung | Speicherung |
| „!0)=?“ | Keine Speicherung | Keine Speicherung |
| „ „ | Keine Speicherung | Keine Speicherung |
| „“ | Keine Speicherung | Keine Speicherung |

Wie erwartet akzeptiert das Programm nur sinnvolle Eingaben.

Auswertung:

Alle Tests ergaben zunächst keine groben Fehler, die nicht sofort behoben werden konnten. Das Produkt gilt also als hauptsächlich fehlerfrei.

**10. Entwicklungsumgebung**

Das Spiel wurde in Java 8 in der IDE Eclipse Oxygen 2.0 auf einem Windows 10 Rechner geschrieben, kompiliert und getestet.

**11. Ergänzungen**

Installationsbedingungen:

Es muss darauf geachtet werden, dass sich alle Bild-Dateien im richtigen Ordner befinden. Außerdem muss die Schriftart „04B\_19\_\_.TTF“ auf dem Rechner installiert sein, damit das Design der Anwendung einheitlich ist.

**12. Scrum-Board**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datum | Aufgaben (to do) | In Arbeit | Testing | Fertig |
| 08.02.18 | Gegner-KI  Scoreboard  FileReader  Menü  Schwierigkeitsgrade | Level-Klasse  Spieler-Movement |  | Texturen  ImageLoader |
| 09.02.18 | Scoreboard  FileReader  Menü  Schwierigkeitsgrade | Gegner-KI | Spieler-Movement | Texturen  ImageLoader  Level-Klasse |