PTP 2020 Projekt: “Snake”

Autoren: Marika Singer und Sarah Guy
Gruppe 5 Donnerstag
Datum der Erstellung: 05.06.2020
Letzte Änderung: 16.08.2020

Sollkonzept-Grobbeschreibung

Projektthema: Snake-Spiel

Worum geht's?

- Spiel
- Schlange, die durch Aufsammeln von Essen wächst
- Highscores, je nach Anzahl des aufgesammelten Essens
- Game Over wenn die Schlange mit sich selbst oder einem Hindernis kollidiert

Wie funktioniert das System?

- Schlange liegt in einem Array der Position und Bild miteinander verknüpft
- Futter wird per Randoms (Positionen) generiert
- sobald ein Futter aufgesammelt wurde werden zwei neue Futter erzeugt, ein GoodFood und ein BadFood
- Aufsammeln wird durch Abgleichen der Koordinaten des Kopfes und des Essens bewirkt
- Array der Schlangenlänge wird beim Aufsammeln des Essens verlängert
- Wenn der Kopf die gleiche Position, wie eines seiner Körperteile hat ist das Spiel vorbei

User Story

Spiel beginnen

Das Spielfeld wird aufgebaut. Die Schlange und das Futter werden erstellt und gezeichnet und für den Spieler im Spielfeld sichtbar.

Schlange kann durch das Bild bewegt werden

Die Schlange lässt sich mittels Pfeiltasten oder Maus durch das Bild bewegen.

Testfälle schreiben
Für die bereits programmierten Klassen bereits erste Testfälle schreiben

Schlange frisst Futter und vergrößert sich dadurch
Wenn der Kopf der Schlange auf ein Futter trifft, frisst sie das Futter und vergrößert sich.

Punktezähler einbauen
Die Punktzahl erhöht sich, wenn die Schlange ein Futter frisst.

Game Over
Game Over Option einbauen. Wenn der Kopf der Schlange auf den Körper trifft und sich die Schlange damit selbst beißt, kommt es zum Game Over.

Andere Arten von Futter hinzufügen
Gutes und Schlechtes Essen hinzufügen, was andere Auswirkungen auf die Schlange hat als das gewöhnliche Futter. Beispielsweise Punktabzug oder andere negative Effekte.

Start-Menu
Ein Startmenü, in dem aus mehreren Optionen ausgewählt werden kann hinzufügen. Das Menü wird als ersten angezeigt, wenn das Spiel gestartet wird. Anschließend kann durch die richtige Menu Auswahl das Spiel begonnen werden.

Music und Sound Optionen
Abspielen von SoundEffekten, wenn die Schlange das Futter frisst.
Hintergrundmusik, die während des Spiels läuft (Achtung! Kann sehr laut sein, wenn man das Spiel das erste Mal laufen lässt. Ist als Default eingeschaltet)

Highscores
Eine Highscore Anzeige, die die erfolgreichsten Spieler und ihre erreichte Punktzahl abbildelt

Aus Zeitmangel haben wir folgende User Stories (noch) nicht umgesetzt:

Portale in neue Welt
Symbole für Portale in neue Welten erstellen. Dort kann der Hintergrund anders sein als in dem Standardspiel oder ein Labyrinth auftauchen.

Labyrinth
Zufälliges Labyrinth erzeugen. Falls die Schlange das Labyrinth berührt, führt dies ebenfalls zu einem Game Over.

Projektplan und Zeitplan

Der Zeitplan und Projektplan wurden im Laufe des Projekts überarbeitet. Die Punkte 15 und 16 konnten wir, wie oben bereits beschrieben, aufgrund des Zeitmangels nicht mehr durchführen. Wir haben die beiden Punkte trotzdem aufgelistet.

Prio	User Story	Zeitaufwand (soll)	Zeitaufwand (ist)
1	Spiel beginnen Spielfeld wird aufgebaut Schlange wird erstellt Futter wird dargestellt	5h	8h
2	Schlange bewegt sich durch Bild -Per Maus -Per Tastaturpfeile	3h	3h (Tastaturpfeile) 3h (Mausklick) → wurde in der endgültigen Version wieder verworfen
3	Testfälle schreiben (nach Test-First-Ansatz)	4h	10h
4	UML Diagramm und GoogleDocs zur besseren Planung erstellen	2h	4h
5	Schlange frisst Futter und vergrößert sich Läuft anschließend normal weiter	2h	2h
6	Punktezähler einbauen (Zählt gefressene Futter mit)	1h	2h

7	Game Over Optionen einbauen: -wenn Schlange sich selbst berührt, verliert der Spieler ebenfalls - eventuelles Speichern von Highscores	2h	1h (wenn Schlange sich selbst berührt)
8	Andere Arten von Futter hinzufügen Schlange ändert Farbe und Form je nach Futter Grafiken erstellen	5h	12h
9	Start-Menu erstellen	6h	8h
10	Music und Sound Optionen	4h	6h
11	Highscores speichern und anzeigen	5h	7h
12	Nachträgliches Refactoring	0h (war nicht eingeplant)	6h
13	Weiteres Testen, Fehlerbehebung und Behandlung von Spezialfällen	4h	6h
14	Sonstiges (Grafiken erstellen und ändern) und Zeitpuffer	5h	10h
15	Portale in neue Welt (Tunnelsymbol) und neue Umgebung einbauen	4h	0h → wurde weggelassen

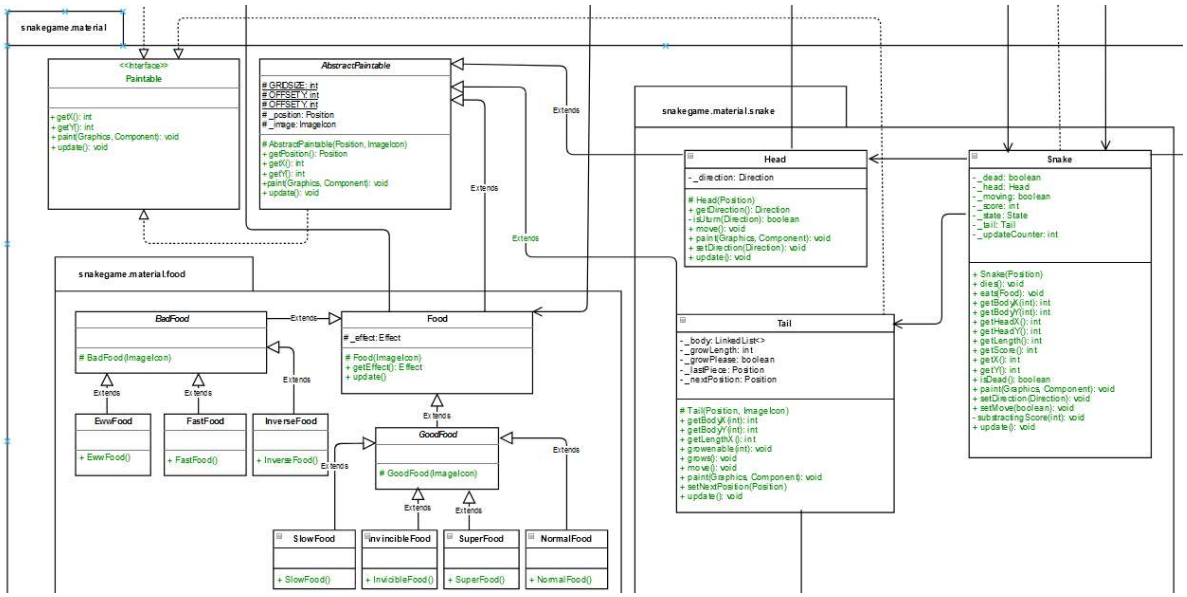
16	Labyrinth einbauen (verhält sich wie GameOver des Spielfelds), wird aber zufällig in einer der neuen Welten erzeugt	4h	0h → wurde weggelassen

Übergreifende, abstrahierte Beschreibung ("Vogelperspektive")

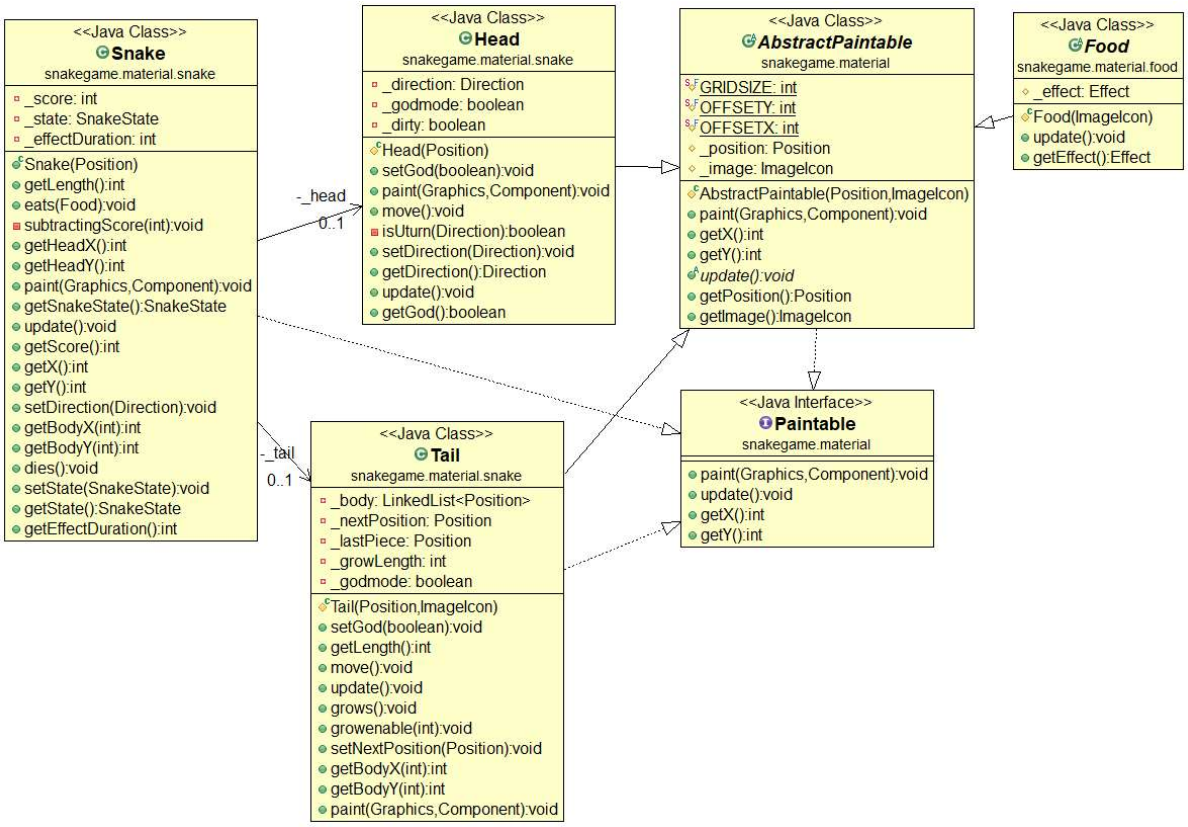
Unser Ziel war es, ein Einfaches, aber ebenfalls einfach erweiterbares Snake Spiel zu programmieren. Unser Schwerpunkt lag dabei klar auf der Funktionalität und der Wiederverwendbarkeit unseres Codes.
Dieses Ziel konnten wir umsetzen. Herausgekommen ist ein Spiel mit Basisfunktionen, das in allen Bereichen einfach erweitert werden kann. Das Menü ist beispielsweise so gestaltet, dass jederzeit neue Menüpunkte hinzugefügt werden können. Auch das Futter der Schlange kann jederzeit um gutes oder böses Essen mit entsprechenden Effekten ergänzt werden. Wir konnten nicht alle Extras und Zusatzfunktionen einbauen, die wir gerne gehabt hätten, aber haben das Grundgerüst für diese Optionen mit unserem Code gelegt.
Hinsichtlich der Programmierkonzepte haben wir versucht, uns möglichst an den in SE2 vermittelten WAM Ansatz (Werkzeug- und Materialansatz) zu halten und die Klassen in entsprechende Packages mit Fachwerten, Material, Werkzeugen und Services aufgeteilt. Das konnten wir auch weitestgehend umsetzen.
Wir hatten geplant, den „Test-First“-Ansatz umsetzen und haben uns beide aber sehr lange gesträubt, uns mit den Tests auseinanderzusetzen. Letztendlich haben wir die Tests nachträglich geschrieben und erst währenddessen festgestellt, dass der Test-First-Ansatz effizienter gewesen wäre, da wir uns dann nicht doppelt mit dem Code beschäftigen hätten müssen. Außerdem hätten wir gerne noch mehr getestet und mehr Testfälle geschrieben, aus Zeitgründen war dies dann nicht mehr möglich.
Wir haben zu Beginn mit draw.io „von Hand“ UML-Diagramme unseres Projekts erstellt, um das Projekt besser planen zu können und eine Übersicht zu erhalten. Dieses haben wir im Verlauf noch ergänzt und verändert. Wir haben und konnten nicht alles genau so umsetzen wie geplant. Die zusätzliche Arbeit hat uns trotzdem sehr dabei geholfen, den Code sinnvoll zu strukturieren. Außerdem konnte uns so, unserer Meinung nach, ein sinnvolles Refactoring gelingen.

UML-Diagramm

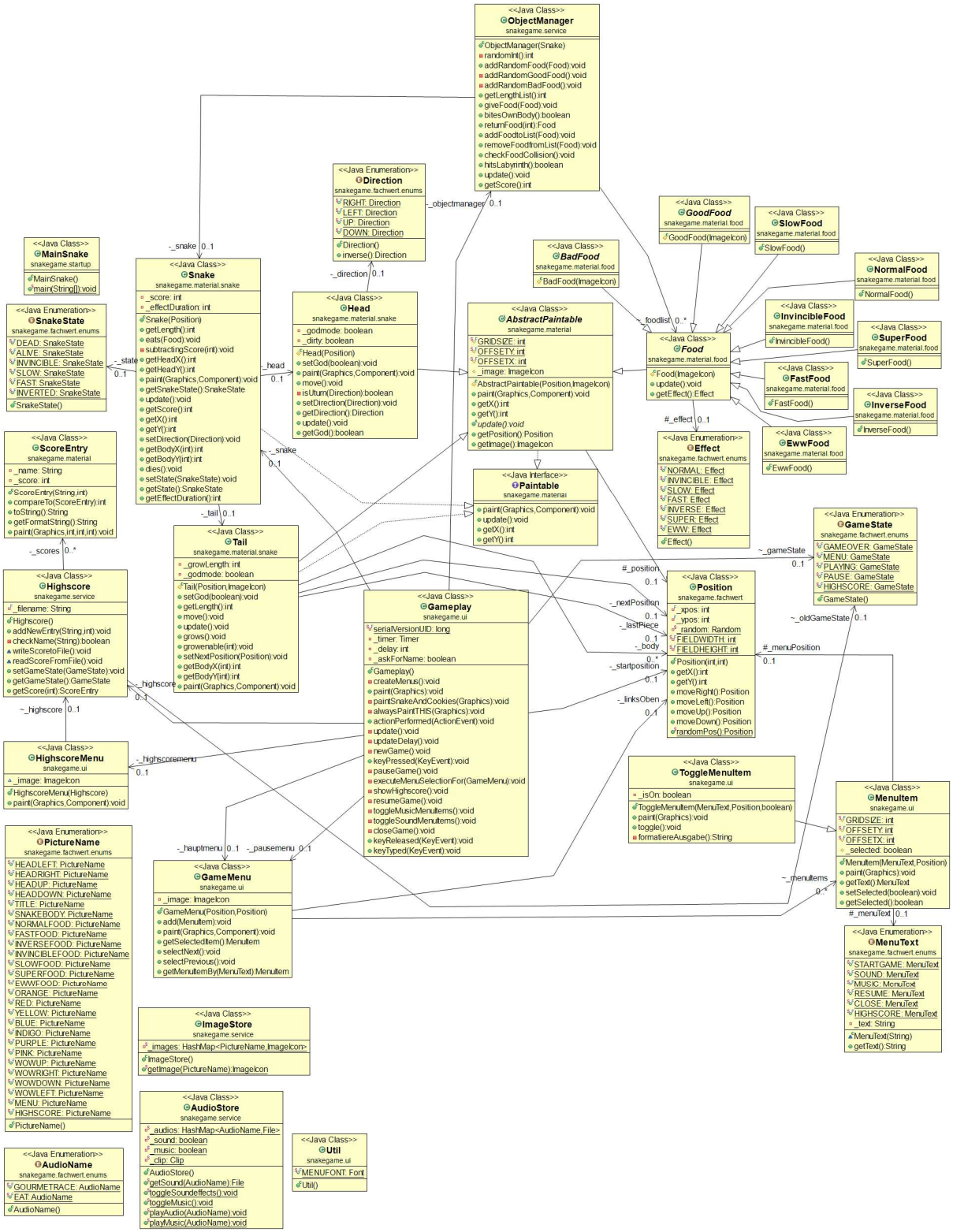
Ausschnitt aus einem der Planungs-UML-Diagramme, die mit draw.io erstellt wurden:



Ausschnitt aus dem finalen Snake-Spiel:



Gesamtübersicht des finalen Snake-Spiels:



(System-)technische Beschreibung

Wie läuft das Zusammenspiel der Klassen?

Hier geben wir kurz eine Übersicht über die wichtigsten Klassen und Interfaces, die wir in unserem Spiel verwenden. Die Liste ist nicht vollständig und soll nur einen groben Überblick geben. Die Klassen liegen nach dem WAM-Ansatz in unterschiedlichen Packages.

- Gameplay
  - Hauptklasse, in der sich das meiste Geschehen abspielt
  - hat Exemplare von der Schlange, Objectmanager, den Menus
  - update (mit einem delay von 100) wird regelmäßig ausgeführt, in update wird paint aufgerufen, so wird das Bild mit jedem update neu gezeichnet
- Snake
  - besitzt ein Exemplar von Head und Tail
  - in ihr wird der Punktestand (Score) gespeichert
  - sie besitzt ebenso einen SnakeState, der festlegt, in welchem Modus sich die Schlange befindet (bsp.:dead, alive, fast, slow,...)
  - einige Effekte, welche einen bestimmten SnakeState auslösen sind zeitlich begrenzt, welche durch einen Counter, welcher sich mit jedem Update herabsetzt, bestimmt sind
  - gibt die x und y Position des Heads an den Tail weiter
  - gibt die Positionen von dem Tail an den Objectmanager weiter
- Head
  - besitzt eine Direction, die festlegt, in welche Richtung die Schlange schaut und bewegt wird, regelt die Bewegung über Position
  - besitzt ein Exemplar von Position
- Position
  - hat eine Methode move(), die eine neue Position zurückgibt
- Objectmanager
  - erhält durch Gameplay die Snake
  - Organisiert das Erscheinen des Foods, sodass jeweils ein GoodFood als auch ein BadFood auf dem Feld liegen
  - managet Kollisionen des Heads mit dem Körper oder mit Food
- AbstractPaintable
  - Abstrakte Klasse, auf die alle Objekte zugreift, die gezeichnet werden (z.B. Schlange, Food)
- Food
  - Oberklasse von allen Arten von Essen (Foods)
  - Erbt von AbstractPaintable, sodass alle Arten von Essen auf die Methoden von AbstractPaintable zugreifen können
- Enums

    - zur Fallunterscheidung z.B., um auf die richtige Grafik aus dem Imagestore zugreifen zu können
- HighscoreMenu
    - Zum Anzeigen des Highscore Menüs
- ScoreEntry
    - Zum Registrieren, Speichern und Vergleichen von Punkten, die während des Spiels erreicht werden

Welche Probleme wurden wie gelöst?

- Die Rotation des Kopfes wird je nach Richtung der zuletzt gedrückten Pfeiltaste festgelegt
- Man kann nicht in die entgegengesetzte Richtung wechseln und so ein versehentliches Game Over bewirken → Bei Keywechsel wird abgefragt, welche Richtung aktiv ist und die entgegengesetzte Richtung wird blockiert

Welche Probleme wurden noch nicht gelöst?

- Aktuell kann Futter noch auf der Schlange auftauchen.

Überlegungen zum Testen

Die Testfälle wurden im Anschluss geschrieben. Als Basis dienten uns folgende Überlegungen:

Testfälle

- PositionTest.java:
    - testet, die Methode moveup() in alle Richtungen von der Klasse Position
    - Randfälle für die Übergänge am Rand auf die andere Seite des Spielfelds sind in jeder Methode an 2. Stelle abgedeckt
- HeadTest.java:
    - testeHeadKonstruktor()
        - überprüft, ob beim Erzeugen eines neuen Heads die Konstruktorwerte übergeben werden
    - testeSetGod()
        - überprüft, ob die Methode setGod() (diese legt den Unbesiegbarkeitsmodus, den man durch den Katzen-Cookie erlangt fest) funktioniert
    - testeMove()
        - hier wird die Position des Heads getestet, nach einem Move in alle vier verschiedene Richtungen (UP, RIGHT, DOWN, LEFT)
    - testeGetDirection()

        - hier wird die Methode setDirection() (welche die Richtung, in die die Schlange schaut festlegt) und getDirection() (der dazugehörige Getter) getestet
        - hierbei wird auch darauf geachtet, dass man nicht in die gegenüberliegende Richtung wechseln kann, in welche die Schlange zu dem Zeitpunkt schaut
        - nach jedem Setten muss zudem auch einmal move() ausgeführt werden, da die Schlange nur einen Richtungswechsel pro Movement erlaubt
- SnakeTest.java
    - testeEats()
        - eine sehr große Testklasse in der Folgendes getestet und abgedeckt wird:
            - es wird getestet, ob der Score sich entsprechend der gegessenen Foods erhöht oder senkt
            - außerdem wird getestet, wie lange ein Effekt anhält und ob der Effektcounter sich verringert bei der update-Methode
            - der Status der Schlange wird abgeprüft
            -

Beschreibung für Benutzer

Wie startet man das Programm?

- Das Programm wird durch das Ausführen der Exe gestartet.
- Zu Beginn wird das Menü mit seinen Menü-Punkten angezeigt.
- Durch das Menü bewegt man sich mithilfe der Pfeiltasten.
- Die markierte Menü-Option wird rot dargestellt.
- Drücken der Enter-Taste führt zur Auswahl und “Aktivierung” der markierten Menü-Option.

Welche Menü-Optionen gibt es und was bewirkt ihre „Aktivierung“?

- Resume: Das bereits begonnene Spiel kann fortgesetzt werden.
- Start Game: Es wird ein neues Spiel gestartet. Wenn man “Start Game” auswählt kann man anschließend durch das Drücken der Pfeiltasten die Schlange in Bewegung setzen
- Sound: Sound beim erfolgreichen Fressen von Nahrung wird abgespielt.
- Music: Hintergrund-Musik des Spiels läuft.
- Highscore: Eine Liste mit den bisher erzielten Highscores wird angezeigt.
- Close: Das Spiel wird geschlossen.

Was ist dann zu tun (Vorgehensweise, Alternativen, Fehler)?

- Mit den Pfeiltasten kann man die Schlange steuern.
- Die Schlange soll auf das gute Futter bewegt werden. Der Spieler muss während des Spiels selbst herausfinden, welches Futter „gut“ und somit zu einem Wachstum der Schlange und zu einer Erhöhung der Punktzahl und zum Eintreten von positiven Effekten führt und welches Futter „böse“ ist und zu negativen Effekten oder Punktabzug führt.
- Alternativ kann auch das Pausenmenü durch Klick der Taste „p“ aufgerufen werden. Somit kann das Pausenmenü aus dem Spiel heraus aufgerufen werden und Optionen wie Sound und Music ausgewählt werden.
- Aktuell gibt es einen Fehler, der noch nicht behoben wurde: Der Keks kann auch auf der Schlange selbst auftauchen. Dies beeinträchtigt den Spielablauf nicht.

Welche Tastenbelegungen gibt es?

- „p”: Pausenstatus → das Pausenmenü wird aufgerufen
- Pfeiltasten: Zur Navigation durch das Menü oder zur Bewegung der Schlange.
- Enter: Zur Auswahl bzw. Bestätigung einer Menü- oder Pausenmenü-Option.
- Leertaste: Zu Beginn hatten wir die Leertaste als Game Over Taste eingefügt. Sie wird aber zu häufig zufällig gedrückt und deshalb haben wir diese Option wieder entfernt. Die Leertaste hat nun keine Funktion mehr.

Fazit und Lessons Learned

Wie wurde die Arbeit aufgeteilt?

Wir hatten in unserer Gruppe zu Beginn Schwierigkeiten die Arbeit aufzuteilen, da wir bei den Absprachen häufig aneinander vorbeigeredet haben und jeder sein “eigenes Süppchen” gekocht hat. Zu Beginn haben wir eher gegeneinander als miteinander gearbeitet. Vieles haben wir zu Beginn doppelt erarbeitet und nicht im Team. Nach einer längeren Aussprache haben wir beschlossen, uns deutlich regelmäßiger über den Stand des Projekts und die genaue Teilaufgabe, an der wir gerade arbeiten gegenseitig zu informieren. Wir haben viel synchron über Discord und geteiltem Bildschirm programmiert und nachdem das Hauptprogramm mit den Basisfunktionen erstellt wurde, konnten wir anschließend die Aufgabenbereiche besser aufteilen. Hierfür haben wir Googledocs genutzt und ein Dokument mit ToDo‘s erstellt, die wir jeweils einem von uns zugeordnet haben. Wenn an einer Aufgabe gearbeitet wurde oder diese fertiggestellt wurde, konnte das sofort in dem Dokument vermerkt werden.

Ein kleiner Einblick in einen der Zwischenstände des Dokuments, in dem wir unseren Fortschritt regelmäßig dokumentiert haben:

Todo:
- Programm fertig schreiben(DONE)
- Grafiken für Schlange fertig machen und schauen, dass die Anzeige-Fehler behoben werden
- User Stories(Done)

- Zeitplan eintragen
- Vortrag vorbereiten (DONE)
- Tests schreiben
- UML (DONE)
- Projektplan und Beschreibung (Maya und Sarah)
- Schnittstellenkommentare (JavaDoc)

Next steps:

- Mausklick (Sarah) (verworfen)
- Graphik ändern (Banner) (Maya)
- Labyrinth (Maya) (verworfen)
- Enemy (Keks) Klasse erstellen (Maya) DONE
- Pausentaste (Sarah) DONE
    - Pause und der Bildschirm gefriert, erneutes Drücken Spiel geht weiter
    - Pause Text (und p für Pausenende)
    - 2 Buttons: Hauptmenu, Spiel fortsetzen
- Menü (Sarah)
    - Spielzustände (Pause, RunningGame, Startmenu, Gameover)
    - Start game
    - *highscore (optional) hashmap oder Liste (Bei Ende vom Spiel kann man eintragen, wie der Name ist → Highscore wird in Liste mit Namen* gespeichert, oder man bricht ab → kein überschreiben)
    - Spiel mit Labyrinth starten
    - → Labyrinth aktivieren?
    - welche Kekse sollen aktiviert sein?
    - Close
    - 5 Knöpfe
- (Mausklick oder Pfeiltasten und Enter)
- Großer Button: Start Game
- Auf alle Fälle: Close Game
- Optionen: Highscore (normaler Button), Labyrinth aktivieren, Kekse aktivieren (als check box)

Status

1. Umorganisation und Sortieren
    - Enemy (Klasse)
    - load Graphics (Klasse)
    - Interface Paintable
    - Schlange aufteilen auf Body und Head
    - Enum für Spielzustände (Sarah)
2. Testen der alten Funktionen
3. neue Funktionen einbauen (neue Kekse und Labyrinth) → Tests &Menü
4. Graphik

Optionales

- Highscore → Verschiedene Namen eingeben
- Musik
- verschiedenes Essen
    - Unbesiegbarkeit
    - langsam/schnell
    - schlechtes Essen
- verschiedene Skins für die Schlange
- Startscreen
- Klassen strukturieren und umstellen

Maya hat deutlich produktiver an dem Projekt gearbeitet, sodass sie deutlich mehr “verwertbaren” Output zum Projekt beigetragen hat. Sarah hat deutlich länger gebraucht und teilweise auch viele Zeitstunden investiert bis dann überhaupt ein verwertbares Ergebnis der Teilaufgaben zustande kam.
Insgesamt hatte das Projekt für beide Teammitglieder einen extrem hohen Workload und eine steile Lernkurve.

Bei den Bearbeitungen der Aufgaben fiel es Maya deutlich leichter als Sarah und sie hat damit deutlich mehr der Funktionen des Spiels programmiert. Maya war trotzdem stets höchst geduldig und hat Sarah stets mit Rat und Tat Beiseite gestanden, sodass wir nun beide auf dem Stand sind, dass wir alle Funktionen unseres Projekts sehr gut verstehen und begründen können, warum wir dieses auf diese Art und Weise umgesetzt haben.

Die Arbeitsaufteilung war grob wie folgt:

Erstellung der Grafiken: Maya
Basisfunktionen des Spiels: Maya und Sarah
Menu: Sarah
UML-Diagramm: Sarah
Dokumentation: Sarah
Food-Optionen: Maya
Music und Sounds: Maya
Highscores: Maya

Endstand

Spiel, welches alle gewünschten Basisfunktionen besitzt und einfach um weitere Optionen erweitert werden kann.

Offene Punkte

Wir hatten noch optional vor ein Labyrinth zu bauen, aber haben uns in diesem Fall lieber darauf konzentriert sauberen Code, der gut weiter verarbeitbar ist zu erstellen.

Ausbaumöglichkeiten und nächste Schritte

Labyrinth, Portale in andere Welten, 2-Playergame, andere Kekse, Time Modus, wahrscheinlich programmieren wir in der Freizeit weiter an dem Spiel.

"Kritische Würdigung" inkl. Workload und Abweichungen vom Zeitplan

Wir sind deutlich vom Zeitplan abgewichen, obwohl wir diesen großzügig geplant hatten. Gründe hierfür waren unsere Programmierkenntnisse, dass wir uns dadurch an Kleinigkeiten aufgehängt haben. Außerdem haben wir viele Doppelarbeiten zu Beginn ausgeführt, weil wir uns schlecht abgesprochen haben. Unerwartete Probleme und Basics haben uns deutlich länger aufgehalten als erwartet.
Uns erging es ähnlich wie anderen Gruppen. Eine sorgfältigere Planung von Beginn an und kein wildes „Losprogrammieren“ hätte unsere Arbeit sehr viel effizienter gemacht.