Multimediasysteme – Abschlussbericht

Team:

Daniel Lovrinovic	k12018625
Lukas Nys	k12110969
Simon Öfferlbauer	k12105213
Stefan Gaderer	k11844600

Thema:

Minispielsammlung in Java mit Ausgabe über die Konsole

Projektbeschreibung:

Unsere Grundidee bestand darin, so viel Multimedia, so einfach wie möglich, darzustellen. Um dies umzusetzen haben wir uns für eine Minispielsammlung entschieden, die jedoch nicht über eine GUI verfügt, sondern bei der die Ausgabe über die Konsole vonstatten geht. Deshalb mussten wir uns zuerst um eine funktionierende Ausgabe der Spiele kümmern und erstellten somit das im Folgenden beschriebene Framework. Ein einzelner Frame wird dabei durch eine Reihe von Zeichen dargestellt. Hier eine zuvor überlegte Beispiel-Ausgabe des Spiels Pong:

Um nun tatsächlich ein Spiel auszugeben wird die Konsole ständig geleert und ein neues Frame mit dem aktuellen Spielstand ausgegeben. Dies funktioniert so schnell, sodass ein interaktives Video simuliert wird. Dazu kommen noch Soundeffekte, die vom jeweiligen Spiel abhängen und selbst produziert und in das Framework eingebettet wurden. Zudem haben wir ein Konstrukt geschaffen, bei dem die gesamte Ausgabe unabhängig vom jeweiligen Spiel funktioniert, das bedeutet, dass jedes Spiel mit seiner eigenen Logik in dieses Framework implementiert werden kann! Die Spiele müssen nur das Interface des Frameworks implementieren und deren Methoden korrekt überschreiben. Somit konnten wir, nachdem das Interface programmiert wurde, parallel am Framework und an den Spielen arbeiten.

Das Hauptprogramm

Grundidee

Da unser Projekt nicht in einer Spiele Engine oder ähnlichem realisiert ist, mussten wir uns um I/O selbst kümmern. Im speziellen: Ausgabe der Bilder an der Konsole (als String), Behandlung von Tastatureingaben, sowie die Ausgabe von Audiofiles über die Lautsprecher. Da diese Funktionalitäten von allen Spielen verwendet werden, wurden diese in einem Hauptprogramm zusammengefasst. Des weiteren definiert das Hauptprogramm den Ablauf der Spiele Selektion, mittels Select-Screen und gibt den Spielen ein einheitliches Äußeres, durch einen dekorativen Rahmen.

Hinweis: Mit Hauptprogramm ist alles gemeint was nicht im Package Games liegt.

Game Interface

```
public interface GameInterface {
   void startGame();
   default void inputReleased(int keyCode){};
   void inputPressed(int keyCode);
   Frame nextFrame();
   String getInstructions();
   String getName();
   default int getScale(){return 1;}
   int millisBetweenFrames();
   boolean hasEnded();
   default int getScore(){return -1;};
}
```

Alle Spiele implementieren das GameInterface, nur darüber kommunizieren diese mit dem Hauptprogramm, dadurch ist die eigentliche Spiele Logik von der Ein- und Ausgabe entkoppelt. Es war das erste Stück Code das von uns entwickelt wurde, somit konnten die Spiele und das Hauptprogramm parallel entwickelt werden.

Erklärung der Methoden:

`void startGame()` Wird vor dem ersten nextFrame aufgerufen. Konkrete funktion vom Spiel abhängig.

`void inputPressed(int keyCode)` Hiermit übermittelt das Hauptprogramm dem Spiel das eine Taste gedrückt wurde. Die Bedeutung der Keycodes ist in JNativHook definiert.

`Frame nextFrame()` Das Herzstück des Interfaces. Hiermit weist das Hauptprogramm dem Game an, den nächsten Frame zu berechnen.

Die Klasse Frame beinhaltet Bilddaten sowie Audiodaten.

```
public class Frame {
	public final ColorChar[][] colorChars;
	public final Sound sound;
	…
```

Die Bilddaten werden in einem 2D-Array von ColorChars gespeichert. Bei ColorChar handelt es sich um eine Datenklasse die aus einem char und einer color besteht. *Hinweis: die color Komponente wird nicht verwendet, es handelt sich um ein nicht implementiertes Feature,*

`default int getScale()` Gibt den Skalierungsfaktor zurück, 1 wenn der Frame unskaliert ausgegeben werden soll.

`int millisBetweenFrames()` Gibt an wie lange zwischen zwei nextFrame Aufrufen gewartet werden soll, bestimmt also die Framerate.

`default int getScore()` Gibt den momentanen Score zurück oder -1 falls das Spiel keinen Score implementiert.

Die restliche Methoden sind selbstsprechend.

Ausgabe der Bilder (Frames)

Ausgangspunkt unseres Programms ist ein .jar File, da die Ausgabe aber über die Konsole erfolgen soll, muss diese erst gestartet werden. Dies passiert in **MainApp.main**, erst wird überprüft ob das Programm momentan in einer Konsole läuft, falls nicht startet es sich selbst in einem neuen Konsolen-Fenster, andernfalls kommt es zum normale Programmablauf.
Idee dazu:
https://stackoverflow.com/questions/7704405/how-do-i-make-my-java-application-open-a-console-terminal-window

Um einen Frame ausgegeben zu können, muss dieser erst in einen druckbaren String konvertiert werden, dazu verfügt die Klasse Frame über eine .toString Methode. Diese funktioniert mit den folgenden Schritten:

- Skalierung des 2D-Arrays, um den Skalierungsfaktor.
- Einrahmung des Frames
- Einfügen des Namens des Spiels in die oberste Zeile. (Sowie des Scores falls vorhanden)
- Umwandlung in String. (Zeile für Zeile)

In der Klasse Screens werden der Titelscreen, GameSelectScreen, GameFinishedscreen und der Instructionscreen gespeichert.

Nachdem ein Frame ausgegeben wurde, muss die Konsole geleert werden, das erfolgt über einen speziellen Clear command in der MainApp.clearSreen Methode.
Über die Utility Klasse InOut.Out passiert die Ausgabe in der Konsole. *Hinweis: diese wurden aus dem Kurs Softwareentwicklung 1 übernommen.*

Tastatureingaben

Um auf Tastatureingaben reagieren zu können müssen diese mit einem Listener abgehört werden. Dies wurde unter Verwendung der JNativHook Bibliothek realisiert, welche das NativKeyListener Interface zur verfügung stellt. Unsere Klasse KeyboardBuffer implementiert dieses Interface. Beim Erstellen wird diese an den GlobalScreen angehängt, welche den Listener bei jedem, Tastenanschlag Informiert.

Bei jedem Tastendruck wird der KeyCode der Taste an eine Liste hinzugefügt. Auf diesen Puffer kann dann, mit ClearBuffer() zugegriffen werden, welche die Liste zurückgibt. und sie leert. Der Vorteil dieses Puffers ist es, dass nicht auf jeden Tastenanschlag sofort reagiert werden muss, sondern die KeyCodes zu einem definierten Zeitpunkt gesammelt verarbeitet werden können.

Ausgabe von Audio

Die Abspielen passiert in mainApp.playAudio einfach mit einem Clip, ganz wie in der Übung demonstriert.
Die dazugehörigen Sounds sind im Package Audio im .wav Format gespeichert, um den Zugriff darauf zu erleichtern, wurde zusätzliche eine Enum Klasse Sound erstellt, welche die Pfade zu den Audiodateien kapselt.

Programmablauf

Der Programmablauf wird durch zwei Funktionen strukturiert, runApp() und runGame(). runApp initialisiert den KeyboardBuffer und zeigt den TitleScreen an. WaitForEnter pausiert den Thread bis Enter gedrückt wird. Dann wird die Konsole geleert und kontinuierlich runGame ausgeführt.

```
public static void runApp() {
    keyboardBuffer = KeyboardBuffer.GetKeyboardBuffer();
    Out.println(Screens.titleScreen());
    waitForEnter();
    clearScreen();
    while(true){
        runGame();
        clearScreen();
        Out.println(Screens.gameFinishedScreen());
        waitForEnter();
        clearScreen();
    }
}
```

runGame fordert den User auf ein Spiel auszuwählen, zeigt ihm die Anleitung und startet das Spiel. DrawFrame und PlayAudio machen genau was man erwarten würde. *waitForNextFrame* pausiert den frame, so lange wie in *GameInterface.millisBetweenFrames* definiert. sendInputs ruft für jeden Tastenanschlag, seid dem letzten Frame, *GameInterface.sendInput() auf.*

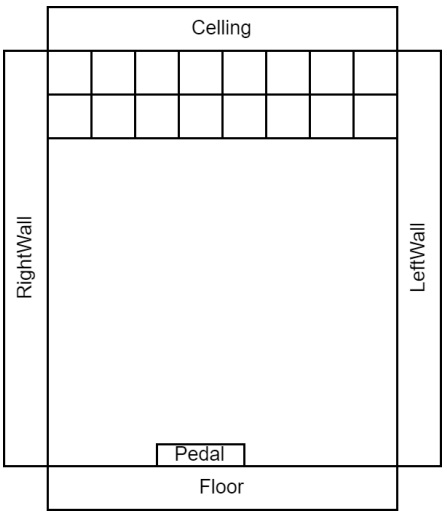
```
private static void runGame()  {
    selectGame();
    showInstructions();
    game.startGame();
    while(!game.hasEnded()){
        Frame frame = game.nextFrame();
        drawFrame(frame);
        playAudio(frame.sound);
        waitForNextFrame();
        sendInputs();
    }
}
```

Breakout

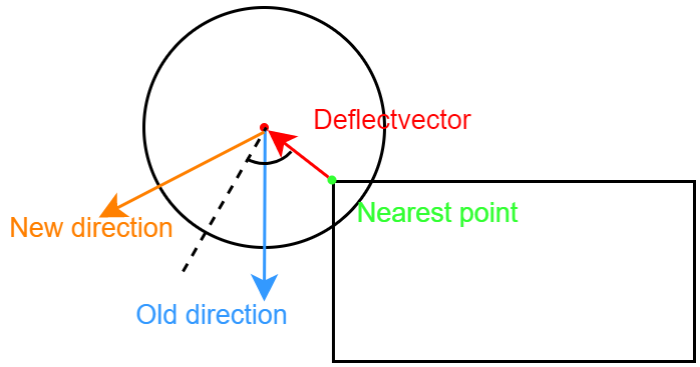
Das Ziel des Arcade Spiel Breakouts ist es alle Blöcke, am oberen Rand des Bildschirms, mit einem beweglichen Ball abzuschießen ohne das er jemals am unteren Rand aufschlägt, er kann dazu mit einem Padal abgewehrt werden.

Klassen

Der Spielball ist ein Objekt der Klasse Ball, alle anderen Spielobjekte, also Wände, Blöcke und das Pedal, sind vom Typ Box.



Ball definiert ein boxCollide(Box b) Methode, die überprüft ob der Ball mit einer Box b kollidiert. Sie wird in jedem Frame, für jede Box überprüft. Tritt eine Kollision auf muss der Ball, reflektiert, also seine Bewegungsrichtung umgekehrt werden. Wie das genau passiert, ist im folgenden Diagramm dargestellt.



Der nearest point ist der Punkt in der Box, der am nähesten, am Mittelpunkt des Balles liegt. Ist die Länge des defelect vectors, kleiner als der Radius des Balles kommt es zur Kollision. Nun muss noch überprüft werden ob sich der Ball auf den nearest point weg oder hinzu bewegt. Wenn er sich hin bewegt, muss die Richtung reflektiert werden. (an der gestrichelten Linie, welche normal zum Deflectvector ist)

Programmablauf

Start

Am Start des Spiels (startGame()), wird eine Liste an zerstörbaren Boxen erstellt.

Pro Frame

Zuerst wird bestimmt in welche Richtung sich das Pedal in diesem Frame bewegen soll, anhand der momentan gedrückten Tasten. Dann wird 10 mal MicroFrame aufgerufen, alle Spielobjekte gezeichnet und der Frame zurückgegeben.

Pro MicroFrame

für jeden, Aufruf von nextFrame wird mehrmals microFrame aufgerufen. Darin werden Pedal und Ball bewegt und dann auf Kollisionen überprüft. Damit ist die “Physikalische” Frame Rate höher als die visuelle. (Die Visuelle ist aufgrund der verwendeten Technologie auf etwa 10-15 fps beschränkt.) Dadurch kann der Ball schneller bewegt werden ohne das es zu fehlerhaften Kollisionen kommt.

Der Ball, sowie das Pedal wird in jedem Microframe bewegt, anschließend werden alle Boxen auf Kollision geprüft.

Wird eine Kollision mit einer zerstörbaren Box detektiert, wird diese gelöscht. (also aus der Liste der entfernt) und es wird der Sound für diesen Frame auf PONG gesetzt.

Wird eine Kollision mit dem Pedal detektiert, wird wird der Sound für diesen Frame auf PONG2 gesetzt. War das Pedal zusätzlich zu diesem Zeitpunkt in Bewegung, wird der Ball in diese Richtung beschleunigt.

Programm Ende

Das Spiel Endet falls die Liste an zerstörbaren Boxen leer ist, oder eine Kollision mit der “Floor” Box detektiert wird.

Snake

Das uns vermutlich allen bekannte Spiel “Snake” wurde auch aufbauend auf dem zuvor beschriebenen Framework ausprogrammiert. Für all jene, denen das Spiel Snake fremd ist hier eine kurze Beschreibung:

Das Spiel Snake handelt von einem Single-Player-Game, welches sozusagen zu den Urgesteinen der Spielewelt zählt. Dieses Spiel lief auf so ziemlich allen Handys der ersten Generation, die dazu in der Lage waren. Der Spieler steuert hierbei eine “Schlange” in einem Spielfeld, auf dem randomisiert Äpfel erscheinen. Mithilfe der Pfeiltasten steuert der Spieler die Schlange in die jeweiligen Himmelsrichtungen und muss somit versuchen die Äpfel zu konsumieren! Wird ein Apfel von der Schlange gefressen, so wächst die Schlange und der Score wird raufgezählt! Ziel ist es einen möglichst hohen Score zu erzielen!

Das Spiel Snake ist prinzipiell sehr simpel zu Programmieren. Man benötigt grob gesagt ein Spielfeld, eine Schlange die die Fähigkeit besitzt zu wachsen und Äpfel die randomisiert auf dem Spielfeld erscheinen.

Spielfeld: Dieses wird durch das zweidimensionale Integer - Array playField dargestellt, welches die Breite width und die Höhe height hat. “int[][] playField = new int[width][height]”

Die Schlange wird als eine ArrayList repräsentiert, in der ein Eintrag einen Punkt (Klasse Point) im Spielfeld wiedergibt. Um das Fortbewegen der Schlange zu simulieren wird in jedem einzelnen Frame das letzte Element der Schlange gelöscht und vorne(abhängig der aktuellen Bewegungsrichtung) eingefügt. Sobald ein Apfel konsumiert wird, wird die Liste um ein Element vergrößert und somit wächst die Schlange.

Die Äpfel werden durch eine einfache Math.random Methode, angewandt auf eine extra x und y Koordinate, berechnet und aufs Spielfeld gesetzt.

Ablauf des Spiels: Das Spiel wird gestartet, es wird ein neues Spielfeld mit den zugehörigen Objekten angelegt. Nun wird auf einen Spieler-Input gewartet. Sobald der Input erfolgt ist bewegt sich die Schlange so lange, bis sie entweder mit sich selbst oder mit der Wand kollidiert. Durch Pfeiltasten-Eingabe des Benutzers werden die Koordinaten X und Y entsprechend auf 1, -1 und 0 gesetzt. Eine Koordinate muss dabei immer 0 sein, um die Bewegung der Schlange optimal zu simulieren. Diese Koordinaten(dirY, dirX) werden daraufhin zu den Koordinaten des vordersten Elements der Schlange hinzu addiert und mit diesen neuen Koordinaten wird ein neues Element der ArrayList hinzugefügt und dann das letzte Element der ArrayList gelöscht. Somit bewegt sich die Schlange. Ob eine Schlange einen Apfel gegessen hat, wird mit der Methode checkCollision(ColorChar[][] chars) überprüft. Hier wird verglichen, ob das Erste Element der Schlange auf der gleichen Position ist, wie der Apfel, sollte dies der Fall sein, dann wird der score um 1 erhöht, genauso wie die Länge der Schlange und es erscheint ein neuer Apfel auf dem Spielfeld. Ob das Spiel aufgrund der zuvor genannten Bedingungen vorbei ist wird durch hasEnded() und snakeBitHerself() realisiert. Zudem werden bei Beginn, beim Konsum eines Apfels und am Ende des Spiels entsprechende Sounds ausgegeben und das Spiel wird abhängig vom Score immer schneller (millisBetweenFrames() werden verkleinert) !

Pong

Ein weiteres Spiele, welches wir implementiert haben, ist das Arcade Spiel Pong. Es wurde als lokales Multiplayer-Game realisiert. In diesem Spiel geht es darum das jeder Spieler den Ball mit dem jeweiligen Paddle im Spielfeld halten soll.

Felder

Die meisten Felder in Pong sind selbsterklärend. Ein Großteil der Variablen sind einfache Integer-Werte. ScoreA und scoreB speichern das Spielergebnis, pongAPos und pongBPos enthalten den unteren Wert des Paddles und pongLength ist die Länge der Paddles. Height und Width speichern die Größe des Spiels und kann geändert werden. Nur die Geschwindigkeit des Balls passt sich nicht der Größe des Spielfeldes an. Das Boolean hasEnded wird zur Beendigung des Games benutzt. Zuletzt besitzt die Klasse noch zwei „Codes“. GameCode speichert einen Integer-Wert zwischen 0 und 2, welcher den Programmablauf nach einem Punkt bestimmt. Zuletzt wurde noch ein SoundCode implementiert. Dieser entscheidet über den Output-Sound nach einem Ereignis.

Methoden und Ablauf des Programmes

Die Methoden hasEnded, getName, getSound und getInstructions werden hier nicht näher erklärt, da ihr Name die Funktion dieser erklärt. StartGame initialisiert alle Felder und erzeugt ein neues Ball-Objekt. Danach kann das erste Bild gefordert werden. Next Frame checkt als erstes den gameCode, ist dieser 2 (im letzten Frame wurde ein Punkt erzielt), wird im nächsten Bild der Spielstand zurückgegeben, und der Code wird dekrementiert, ist er 1 wartet der Thread vier Sekunden, dekrementiert den Integer-Wert ein weiteres Mal und setzt den Frame zurück. Bei dem Wert 0 wird das Spiel normal fortgesetzt. Zuerst wird die Methode moveBall abgerufen. Hier wird nach dem Testen bestimmter Bedingungen, die nächste Bewegung des Balls bestimmt. Danach wird das Bild „gezeichnet“ und der Frame zurückgegeben. Zu guter Letzt kommen wir zur Methode moveBall. Diese bestimmt den Sound, ändert den Spielstand und bewegt den Ball in die richtige Richtung. Zuerst wird getestet ob die nächste Bewegung des Balls außerhalb des Feldes wär, falls ja wird der soundCode auf 1 gesetzt und ball.hitBorder ausgeführt, als nächste ob der Ball hinter einem Spieler wäre und ob der Spieler den Ball trifft. Falls ein Punkt erzielt wurde, erhält der andere Spieler einen Punkt und der Sound- und GameCode werden auf 2 gesetzt, sonst nur der SoundCode auf 0 gesetzt. Am Schluss wird der Ball bewegt.

Klasse Ball

Um das Programm etwas übersichtlicher zu machen, wurde eine Zusatzklasse implementiert, welche die Bewegung des Balles simuliert. Hier sind Felder für die aktuelle Position und den jetzigen Richtungsvektor abspeichert. Der Ball startet in der Mitte des Feldes. Der erste Bewegungsvektor besteht vx, welcher nur der Difficulty-Faktor ist, und vy, welcher eine Zufällige Nummer zwischen 0.5 und 1 multipliziert mit der Schwierigkeit ist. Des Weiteren enthält die Klasse die Methoden, move (addiert den Vektor zur Position), setDiff (ändert die Schwierigkeit) und hitPlayer beziehungsweise hitBorder. Diese ändern den Vektor für das gegebene Ereignis.

Tetris

Tetris zählt wohl zu den bekanntesten Videospielen der Welt, in dem mit verschiedenen

sogenannten "Tetrominos", eine Reihe gebildet werden muss um Punkte zu erzielen. Eine vollständige Reihe wird hierbei wieder aus dem Spielfeld entfernt. Das Spiel endet sobald die "Tetrominos" bis zum oberen Spielfeldrand reichen.

Obwohl das Spielprinzip einfach ist hat sich die Implementierung schwieriger als gedacht herausgestellt. Die Zeichnung des Spielfelds schlägt unregelmäßig fehl und da die Logik des Spiel auf diesem basiert folgen daraus einige Fehler.

Spielfeld:

Das Spielfeld ist ein zweidimensionales Array aus Cholorchars in dem die Blöcke gezeichnet werden. So kann das aktuelle Spielfeld gespeichert und mit diesem interagiert werden.

Blöcke:

Die verwendeten Formen sind in einem dreidimensionalen Cholorchar Array gespeichert so erfolgt der zugriff auf die verschiedenen Formen über einen Zufalls Parameter, der über eine gemischte Liste erzeugt wird. Es werden sieben verschiedene Formen verwendet.

Steuerung:

Für die Steuerung der Blöcke werden diese in dem Array verschoben, wobei sie an der alten Stelle gelöscht und an der neuen gezeichnet werden. Hierbei muss überprüft werden ob sie mit anderen Formen oder dem Spielfeldrand kollidieren würden, und so entsprechend reagiert werden, also die Bewegung verhindern oder dort verankern und einen neuen Block generieren. Ursprünglich angedacht war es die Formen noch rotieren lassen dies ist aber in der aktuellen Version nicht möglich.

Spielablauf:

Zu Beginn des Spiels wird ein neues Spielfeld und der erste Block generiert. Dieser bewegt sich in regelmäßigen Abständen nach unten bis er mit der Spielfeldgrenze kollidiert und dort verankert wird. Anschließend wird ein neuer Block, mit der newPiece() Methode erzeugt.

Das neuzeichnen des gerade aktiven Blockes wird in Draw() geregelt. So wird in dieser mit collidesAt überprüft ob ein Bewegen zur neuen Position überhaupt möglich ist und falls diese Kollision an der oberen Spielfeldgrenze passiert wird das Spiel beendet. Die aktuellen Koordinaten des aktiven Stückes wird in einem int Array gespeichert so wird es in draw() an der alten Stelle aus dem Spielfeld gelöscht und an der neuen Position eingefügt. Wird eine komplette Reihe gebildet wird diese entfernt und der Punktestand wird erhöht.

Dieser Ablauf wird fortgesetzt bis das Spiel endet.

Zusammenfassung und Erfolgsbericht

Um abschließend noch einmal einen Bezug zwischen unserem Projekt und Multimedia herzustellen gibt es hier noch eine kleine Zusammenfassung:

Zuerst wurde das Interface “GameInterface” erstellt, auf welchem jedes Spiel aufgebaut wird. Somit konnte das Framework und die Spiele parallel entwickelt werden. Die Ausgabe der Spiele erfolgt durch ständiges updaten der Konsole, wobei die Konsole geleert und der neue Frame ausgegeben wird. Die Spiele werden als Zusammensetzung von Zeichen repräsentiert, welche dann durch ständiges neu-ausgeben ein interaktives Video darstellen sollen. Audio wird auch über das Framework ausgegeben, der Zeitpunkt der Ausgabe wird jedoch in der Logik der Spiele implementiert. Zusammenfassend ist es uns also gelungen mit unserem Projekt die 4 Gebiete Audio, Video, Bild und Text zu verarbeiten und zu verwenden.

Wir haben durch etwas mehr Zeitaufwand insgesamt 4 Spiele mithilfe des Frameworks, welches die meiste Zeit in Anspruch genommen hat, programmieren und ausgeben können. Zudem wurde eine eigene kleine Soundbibliothek erstellt.
Die Implementierung der Komponente Color in ColorChar ist uns aufgrund zeit technischer Gründe nicht mehr gelungen!

Wir hoffen Ihnen hat unser Projekt gefallen und vielleicht konnten wir Sie für Ihr eigenes Projekt motivieren!

Mit freundlichen Grüßen,

Ihr Projektteam!

Quellen:
https://github.com/kwhat/jnativehook
https://zetcode.com/javagames/snake/
https://en.wikipedia.org/wiki/Breakout_(video_game)
https://en.wikipedia.org/wiki/Tetris
https://en.wikipedia.org/wiki/Pong