Fischgame

Linus Wigger
26.3.2014
Milestone II

Inhaltsverzeichnis

1. Projektbeschreibung

Ziel des Projekts ist es, ein Computerspiel zu erstellen. Als Vorlage dient das Flashgame Fishy (Spielbar unter *http://www.supergames.ch/online-game/fishy/118-spielen*). Der Spieler steuert einen anfangs kleinen Fisch und von der Seite kommen andere Fische. Ziel des Spiels ist es, den grösseren Fischen auszuweichen und die kleineren Fische aufzufressen. Frisst der Spieler andere Fische, wächst er selbst, so dass er mit der Zeit auch grössere Fische fressen kann. Die Abbildung unten verdeutlicht dies. Der rote Fisch ist der Spieler. Er kann sich innerhalb des hellblauen Bereichs bewegen. Die Gegner kommen aus den violetten Bereichen am Bildrand.
Beim Projekt Fischgame soll dieses Spiel mithilfe von Gamegrid nachgebaut werden.
Das Spiel wurde ausgewählt, weil es einen einfachen Aufbau hat, der von Grund auf neu programmiert werden kann. Ausserdem ergeben sich viele Möglichkeiten zur Erweiterung des Basisspiels, etwa das Hinzufügen neuer Fischarten und komplexeren Bewegungsmustern für die computergesteuerten Fische.

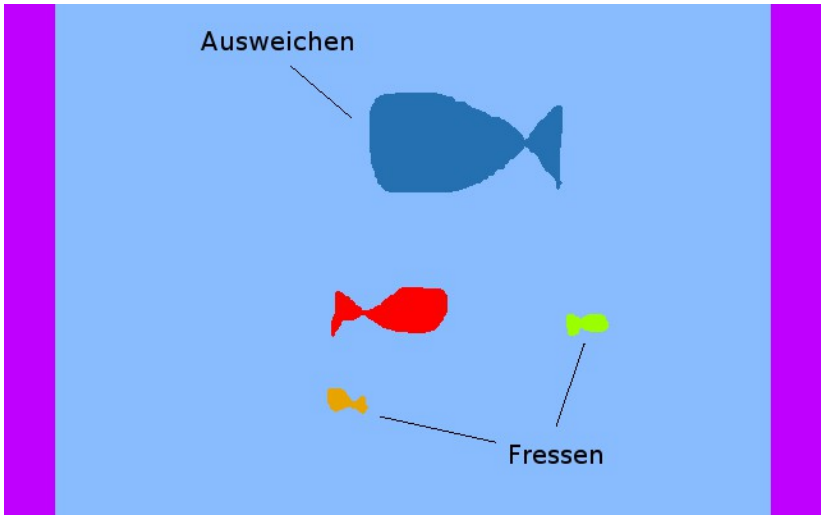

Abbildung 1: Skizze des Spiels

2. Problemdefinition

Für das Erstellen von Fischgame wird Folgendes benötigt:

-Fische, die sich von selbst bewegen
-Fisch, der durch Input vom Spieler gesteuert wird
-Kollisionserkennung bei Fischen
-Der Fisch des Spielers muss wachsen können

Ebenso sind die folgenden Punkte zu beachten. Sie sind nicht absolut notwendig für das Funktionieren des Spiels, aber zumindest die oberen bewirken, dass das Spiel überhaupt Spass macht:

-Zufallselemente, z.B. Wo Computerfische auftauchen, wie gross sie sind etc.
-Bestenliste, die zwischen Spielen gespeichert wird
-Spielmenü
-Mehr als nur eine Fischsorte(z.T. noch mit speziellen Eigenschaften)
-Verschiedene Levels

3. Anforderungsanalyse

Die bei der Problemdefinition genannten Kriterien lassen sich mit Java alle erfüllen. Einzig der Punkt „Der Fisch des Spielers muss wachsen können“ stellte sich heraus, dass er sich nur lösen liess, indem das Wachstum in Stufen statt stetig erfolgt.

4. Spezifikation

Benötigt werden garantiert:

-Klasse für das Spiel selbst, Unterklasse von Gamegrid
-Klasse für den Fisch des Spielers, muss auf Tastendruck reagieren, Unterklasse von Actor
-Klasse für Computerfische, Unterklasse von Actor

Vorzugsweise hat es auch:

-Klasse zur Tastatureingabe, falls kein Spielerfisch anwesend ist
-Superklasse für den Spielerfisch und die Computerfische

Use Cases:

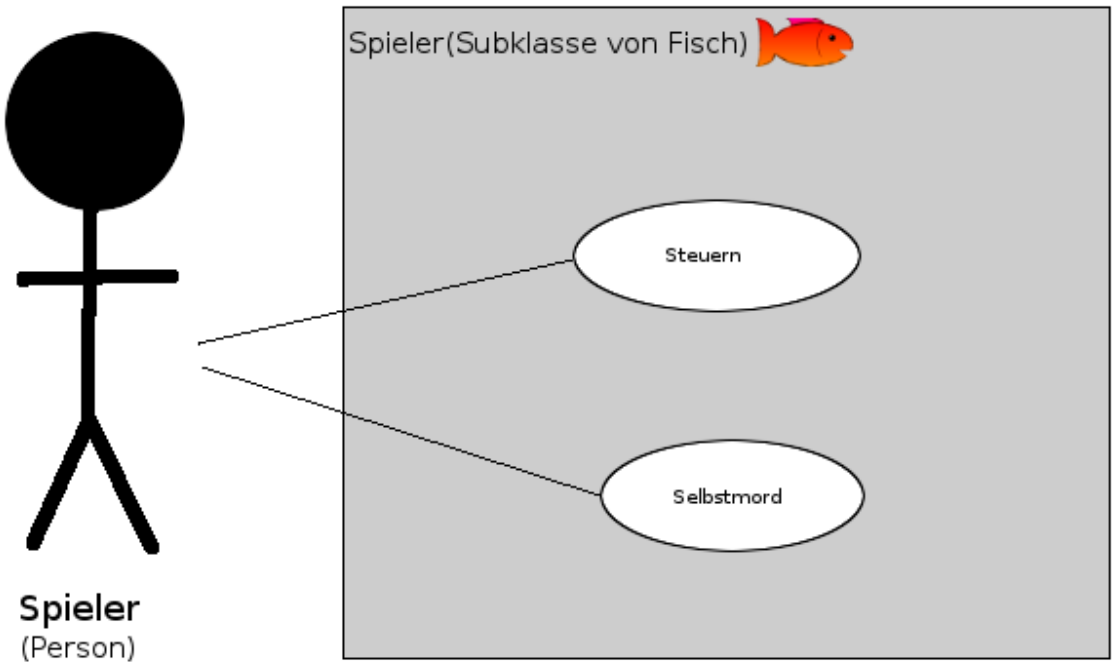

Abbildung 2: Interaktion von Spieler und Fisch

Die Abbildung 2 zeigt, was der Spieler mit seinem Fisch machen kann. Mit den Pfeiltasten kann er den Fisch Steuern. Ausserdem sollte er eine Möglichkeit haben, das Spiel zu beenden, wenn der Spielerfisch grösser ist als alle anderen Fische und so nicht mehr gefressen werden kann (womit das Spiel beendet wäre). Die Abbildung 3 zeigt die Interaktion des Spielers mit dem Eingabefeld für die Bestenliste. Wenn der Spieler ein Zeichen eingibt, werden einige Zeichen nicht zugelassen, hauptsächlich damit in der Bestenliste kein „Name“ auftaucht, der „``]§“ oder so lautet. Der Spieler kann auch ein Zeichen löschen, falls ein falsches eingegeben wurde und schliesslich kann er seinen Namen bestätigen, um die Eingabe abzuschliessen.

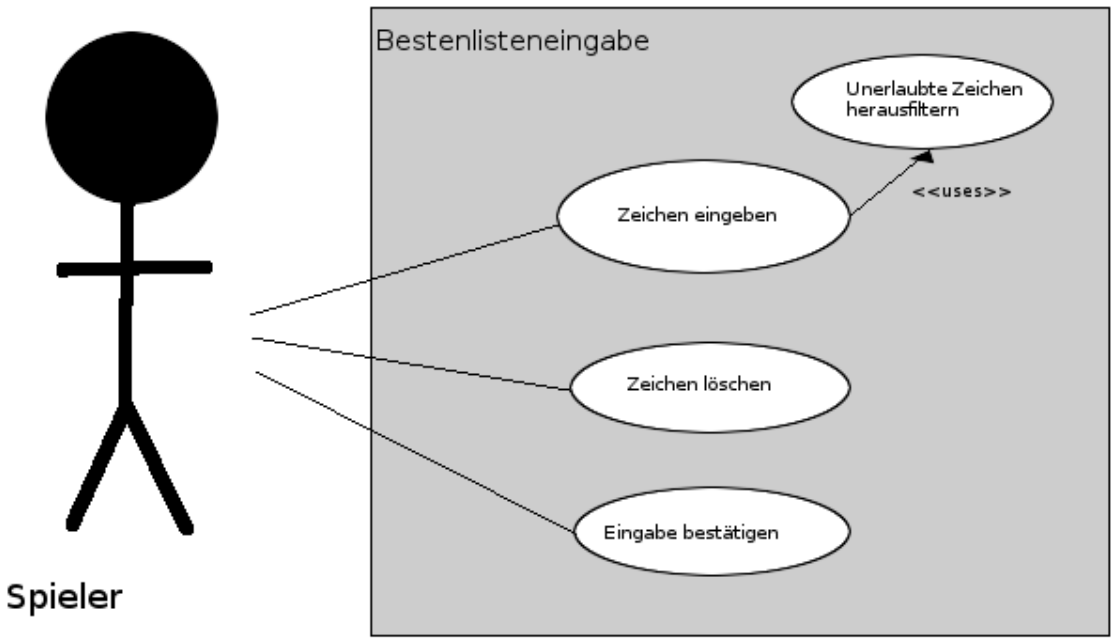

Abbildung 3: Interaktion von Spieler und Bestenlisteneingabe

5. Entwurf

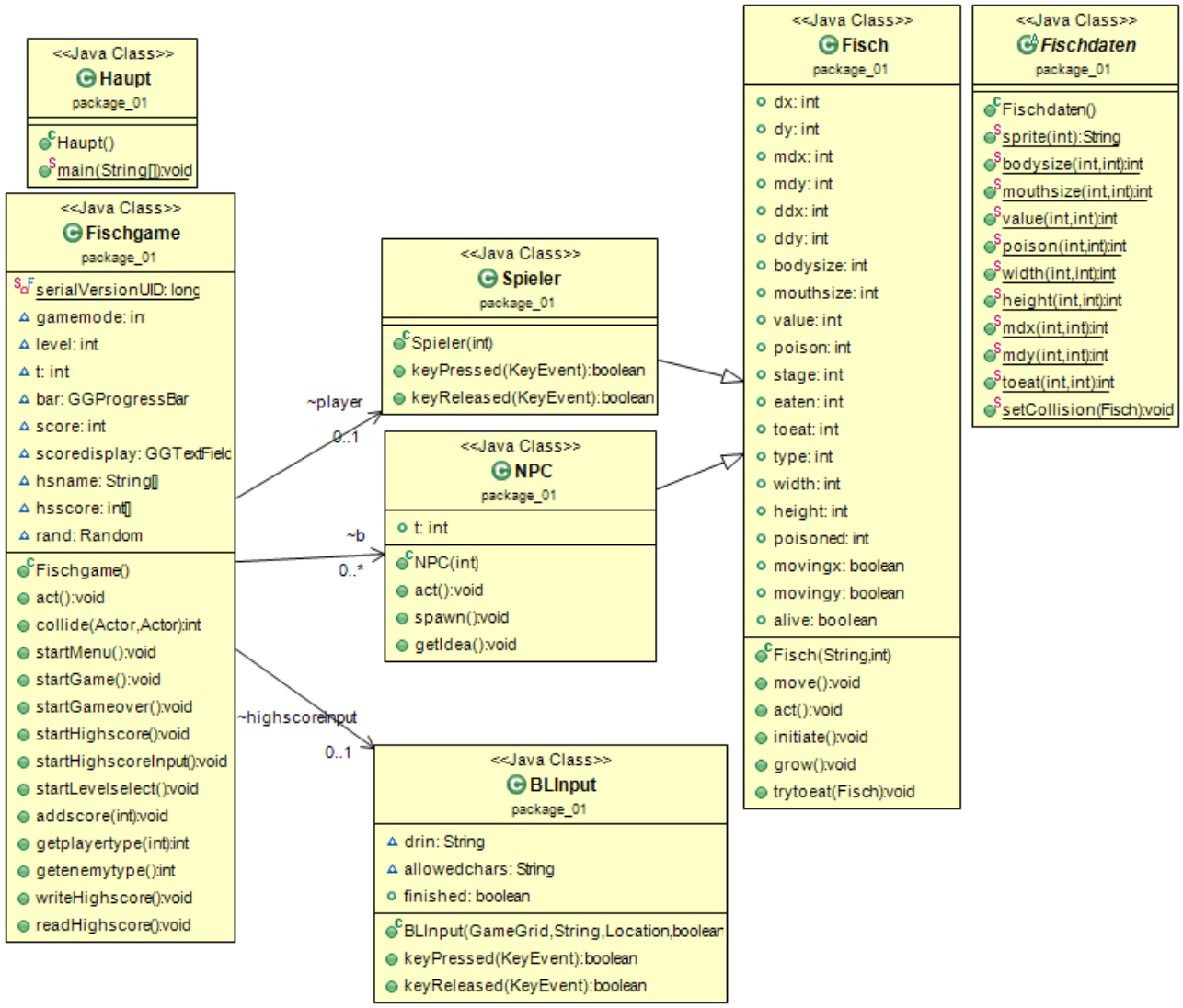

Abbildung 4: UML-Diagramm aller Klassen von Fischgame

In der Abbildung 4 sind alle Klassen von Fischgame ersichtlich. Die Klasse Haupt startet das Spiel, nämlich die Klasse Fischgame, die alles steuert. Die Klassen Spieler (Fisch des Spielers) und NPC (Computerfische) sind beides Unterklassen der Klasse Fisch. Die abstrakte Klasse Fischdaten existiert, um der Klasse Fisch (und ihren Unterklassen) die Eigenschaften für verschiedene Fischarten zu liefern. Die letzte Klasse, BLInput, wird verwendet, um die Namenseingabe für die Bestenliste durchzuführen.

6. Implementierung

```
package package_01;

/* Datum:          24.03.2014
Projektname:       Fischgame
Name:              Linus Wigger
Hauptquellen:      http://www.gamegrid.ch/ (Dokumentation von gamegrid-Klassen),
http://stackoverflow.com/ (Probleme beim Programmieren)
*/
//Klasse Haupt: Startet das Spiel

public class Haupt {
      public static void main(String[] args) {
            new Fischgame();
      }
}
```

```
package package_01;

import java.awt.Color;
import java.io.*;
import java.util.Random;

import ch.aplu.jgamegrid.*;

/* Klasse Fischgame: Das meiste passiert hier
Methoden:
act() - Prüft, ob bestimmte Fische(z.B. der Spieler) gefressen wurden und
handelt entsprechend (neue Fische hinzufügen, Spielmodus wechseln)
collide() - Macht, dass zwei sich berührende Fsiche versuchen, einander zu
fressen
addscore() - Fügt dem Punktetotal die angegebene Menge hinzu und zeigt die
Punkte an
startMenu(), startGame(), startGameover(), etc. - Entfernt alles und fügt je
nach Modus verschiedene neue Fische hinzu
writeHighscore() - Speichert die aktuelle Bestenliste im File
fischgame_highscore.txt
readHighscore() - Liest die Daten aus fischgame_highscore.txt heraus
*/
public class Fischgame extends GameGrid implements GGActorCollisionListener {
      private static final long serialVersionUID = 7742411446668213885L; //Keine
Ahnung, wofür das ist

      int gamemode;     //0: Menü  -  1: Spiel  -  2: Gameover  -  3: Highscore
-  4: Highscore-Input - 5: Levelauswahl
      int level;        //0: Klassisch  -  1: Erweitert
      int t;
    Spieler player;
      NPC [] b = new NPC[5];
      GGProgressBar bar;
      BLInput highscoreinput;
      int score;
      GGTextField scoredisplay;
      String [] hsname =
{"Hai","Hecht","Wal","Eisbär","Krokodil","Tintenfisch","Pinguin","Frosch","Seepf
erdchen","Krebs"};
      int [] hsscore = {1000,900,800,700,600,500,400,300,200,100};
      Random rand = new Random();

      //Wichtige Methoden
```

```
public Fischgame()
{
  super(511, 511, 1, null, "sprites/BG_1.png", false);

  setTitle("Fischgame");
  try {
          readHighscore();
    } catch (IOException e) {
          e.printStackTrace();
    }

  startMenu();


  setSimulationPeriod(40);
  show();
  doRun();
}

public void act(){
      switch (gamemode){
      case 0:
            if (!b[0].alive){
                  startLevelselect();
            }
            if (!b[1].alive){
                  dispose();
            }
            if (!b[2].alive){
                  startHighscore();
            }
            break;
      case 1:
            for (int i=0; i < b.length; i++){
                  if (b[i].alive==false){
                        b[i]=new NPC(getenemytype());
                        addActor(b[i], new Location(1000, 1000));
                        b[i].spawn();
                        player.addCollisionActor(b[i]);
                  }
                  if (!player.alive){
                        startGameover();
                  }
                  if (player.poisoned > 0){
                        bar.setBgColor(new Color(63,255,63));
                        bar.setStripColor(new Color(0,128,0));
                        bar.setValue(player.eaten);
                        scoredisplay.setBgColor(new Color(63,255,63));
                  } else {
                        bar.setBgColor(new Color(63,127,255));
                        bar.setStripColor(new Color(0,0,255));
                        bar.setValue(player.eaten);
                        scoredisplay.setBgColor(new Color(63,127,255));
                  }
            }
            break;
      case 2:
            if (b[1].getX() < 4){
                  for (int i=0; i<10; i++){
                        if (score > hsscore[i]){
```

```
                                    startHighscoreInput();
                                    break;
                              }
                        }
                        if (score <= hsscore[9]){
                              startMenu();
                        }
                  }
                  break;
            case 3:
                  if (!b[0].alive){
                        startMenu();
                  }
                  if (!b[1].alive){
                        hsname[0] = "Hai";
                        hsscore[0] = 1000;
                        hsname[1] = "Hecht";
                        hsscore[1] = 900;
                        hsname[2] = "Wal";
                        hsscore[2] = 800;
                        hsname[3] = "Eisbär";
                        hsscore[3] = 700;
                        hsname[4] = "Krokodil";
                        hsscore[4] = 600;
                        hsname[5] = "Tintenfisch";
                        hsscore[5] = 500;
                        hsname[6] = "Pinguin";
                        hsscore[6] = 400;
                        hsname[7] = "Frosch";
                        hsscore[7] = 300;
                        hsname[8] = "Seepferdchen";
                        hsscore[8] = 200;
                        hsname[9] = "Krebs";
                        hsscore[9] = 100;
                        try {
                              writeHighscore();
                        } catch (IOException e) {
                              e.printStackTrace();
                        }
                        startHighscore();
                  }
                  break;
            case 4:
                  if (highscoreinput.finished){
                        for (int i=0; i<10; i++){
                              if (score > hsscore[i] &&
highscoreinput.drin.length()>0){
                                    for (int j = 9; j > i; j--){
                                          hsname[j]=hsname[j-1];
                                          hsscore[j]=hsscore[j-1];
                                    }
                                    hsname[i]=highscoreinput.drin;
                                    hsscore[i]=score;
                                    try {
                                        writeHighscore();
                                    } catch (IOException e) {
                                        e.printStackTrace();
                                    }
                                    break;
                              }
                        }
```

```
                        startMenu();
                }
                break;
        case 5:
                if (!b[0].alive){
                        level=0;
                        startGame();
                }
                if (!b[1].alive){
                        level=1;
                        startGame();
                }
                break;
        }
        t++;
  }

public int collide(Actor actor1, Actor actor2) {
      Fisch m = (Fisch) actor1;
      Fisch n = (Fisch) actor2;
      m.trytoeat(n);
      if (n.alive==false){
            bar.setMax(player.toeat);
            bar.setValue(m.eaten);
            addscore(n.value);
            removeActor(n);
      } else{
      n.trytoeat(m);
      if (m.alive==false){
            bar.setMax(player.toeat);
            bar.setValue(m.eaten);
            removeActor(m);
      }
      }
      return 0;
}

//Start[Gamemode]

public void startMenu(){
      gamemode=0;
      removeAllActors();
      t=0;
      score=0;

    player = new Spieler(0);
    player.grow();
    addActor(player, new Location(255,255));
    addKeyListener(player);

b[0] = new NPC(-1);
b[0].mdx = 0;
addActor(b[0], new Location(100, 100));
player.addCollisionActor(b[0]);

b[1] = new NPC(-1);
b[1].mdx = 0;
    b[1].grow();
addActor(b[1], new Location(411, 411));
player.addCollisionActor(b[1]);
```

```
      b[2] = new NPC(-1);
      b[2].mdx = 0;
          b[2].grow();
          b[2].grow();
      addActor(b[2], new Location(100, 411));
      player.addCollisionActor(b[2]);

          for (int i=3; i < b.length; i++){
            b[i]= new NPC(-1);
            addActor(b[i], new Location(1, 1000));
            b[i].mdx=0;
            player.addCollisionActor(b[i]);
          }

          bar = new GGProgressBar(this, new Location(0,-50), 500, 10);
          scoredisplay = new GGTextField(this, "Punkte: 0", new Location(30,-
10), true);

          player.addActorCollisionListener(this);
      }

      public void startGame(){
            gamemode=1;
            removeAllActors();
            t=0;
            score=0;

          player = new Spieler(getplayertype(level));
          addActor(player, new Location(255,255));
          addKeyListener(player);

          for (int i=0; i < b.length; i++){
            b[i]= new NPC(getenemytype());
            addActor(b[i], new Location(1, 1));
            b[i].spawn();
            player.addCollisionActor(b[i]);
          }

          bar = new GGProgressBar(this, new Location(255,500), 500, 10);
          bar.setTextColor(TRANSPARENT);
          bar.setBgColor(new Color(63,127,255));
          bar.setStripColor(new Color(0,0,255));
          bar.setMin(0);
          bar.setMax(player.toeat);
          bar.setValue(0);

          scoredisplay = new GGTextField(this, "Punkte: 0", new Location(30,10),
true);
          scoredisplay.setBgColor(new Color(63,127,255));
          scoredisplay.setTextColor(BLACK);
          scoredisplay.show();

          player.addActorCollisionListener(this);
      }

      public void startGameover(){
            gamemode=2;
            removeAllActors();
            t=0;

          player = new Spieler(0);
```

```
        addActor(player, new Location(1,-50));

      b[0] = new NPC(-1);
      b[0].dx = -1;
          for (int i=0; i<5; i++){
            b[0].grow();
          }
      addActor(b[0], new Location(511, 255));

      b[1] = new NPC(-1);
      b[1].dx = -1;
          for (int i=0; i<6; i++){
            b[1].grow();
          }
      addActor(b[1], new Location(611, 255));

          for (int i=2; i < b.length; i++){
            b[i]= new NPC(-1);
            addActor(b[i], new Location(1, 1000));
            b[i].mdx=0;
          }

          bar = new GGProgressBar(this, new Location(0,-50), 500, 10);
          scoredisplay = new GGTextField(this, "Punkte: 0", new Location(30,-
10), true);
       }

      public void startHighscore(){
            gamemode=3;
            removeAllActors();
            t=0;
            score=0;

          player = new Spieler(0);
          addActor(player, new Location(100,100));
          addKeyListener(player);
          player.grow();

      b[0] = new NPC(-1);
      b[0].mdx = 0;
          b[0].grow();
          b[0].grow();
          b[0].grow();
      addActor(b[0], new Location(100, 411));
      player.addCollisionActor(b[0]);

      b[1] = new NPC(-1);
      b[1].mdx = 0;
          b[1].grow();
          b[1].grow();
          b[1].grow();
          b[1].grow();
      addActor(b[1], new Location(411, 411));
      player.addCollisionActor(b[1]);

          for (int i=2; i < b.length; i++){
            b[i]= new NPC(-1);
            addActor(b[i], new Location(1, 1000));
            b[i].mdx=0;
            player.addCollisionActor(b[i]);
          }
```

```
            try {
                    readHighscore();
              } catch (IOException e) {
                    e.printStackTrace();
              }

            bar = new GGProgressBar(this, new Location(0,-50), 500, 10);
            scoredisplay = new GGTextField(this, "", new Location(200,-10), true);

            for (int i=0;i<10;i++){
              GGTextField hsspot = new GGTextField(this, " " + hsname[i] + " :    "
+ hsscore[i] + " ", new Location(195,100+ 25*i),true);
              hsspot.setBgColor(new Color(75+20*i,255,75+20*i));
              hsspot.setTextColor(new Color(180-20*i,0,180-20*i));
              hsspot.show();
            }

            player.addActorCollisionListener(this);
      }

      public void startHighscoreInput(){
            gamemode=4;
            removeAllActors();
            t=0;

      GGTextField sometext = new GGTextField(this, "Mit deiner Punktzahl von " +
score + " hast du die Bestenliste erreicht!", new Location(100,100),true);
      sometext.setBgColor(new Color(63,127,255));
      sometext.show();

      GGTextField moretext = new GGTextField(this, "Gib bitte deinen Namen ein:
", new Location(100,128),true);
      moretext.setBgColor(new Color(63,127,255));
      moretext.show();

            highscoreinput = new BLInput(this, "Fritzi", new Location(264,128),
true);
            addKeyListener(highscoreinput);
      }

      public void startLevelselect(){
            gamemode=5;
            removeAllActors();
            t=0;

            player = new Spieler(0);
            player.grow();
            addActor(player, new Location(255,255));
            addKeyListener(player);

      b[0] = new NPC(-2);
      b[0].mdx = 0;
      addActor(b[0], new Location(100, 100));
      player.addCollisionActor(b[0]);

      b[1] = new NPC(-2);
      b[1].mdx = 0;
          b[1].grow();
      addActor(b[1], new Location(411, 411));
      player.addCollisionActor(b[1]);
```

```
        for (int i=2; i < b.length; i++){
          b[i]= new NPC(-1);
          addActor(b[i], new Location(1, 1000));
          b[i].mdx=0;
          player.addCollisionActor(b[i]);
        }

        bar = new GGProgressBar(this, new Location(0,-50), 500, 10);
        scoredisplay = new GGTextField(this, "Punkte: 0", new Location(30,-
10), true);

        player.addActorCollisionListener(this);
    }

    // Verschiedene Methoden

    public void addscore(int amount){
          String foo = "Punkte: ";
          score+=amount;
          foo = foo + score;
          scoredisplay.setText(foo);
    }

    public int getplayertype(int map){
          switch(map){
          case 0:
                return 0;
          case 1:
                return 2;
          }
          return 1;
    }

    public int getenemytype(){
          double decision=rand.nextDouble();
          switch(level){
          case 0:
                return 1;
          case 1:
                if (decision < 0.65){
                      return 3;
                } else if (0.65 <= decision && decision <= 0.8) {
                      return 4;
                } else if (0.8 <= decision && decision <= 0.85) {
                      return 5;
                } else if (0.85 <= decision && decision <= 1) {
                      return 6;
                }
          }
          return 1;
    }

    public void writeHighscore() throws IOException{
        FileWriter fw = new FileWriter("fischgame_highscore.txt");
        BufferedWriter bw = new BufferedWriter(fw);
        for (int i=0;i<10;i++){
          bw.write(hsname[i]);
          bw.newLine();
          bw.write(Integer.toString(hsscore[i]));
          bw.newLine();
```

```
            }
            bw.close();
        }

        public void readHighscore() throws IOException{
            FileReader fr = new FileReader("fischgame_highscore.txt");
            BufferedReader br = new BufferedReader(fr);

            for (int i=0;i<10;i++){
              hsname[i]=br.readLine();
              hsscore[i]=Integer.parseInt(br.readLine());
            }

            br.close();
        }
}
```

```
package package_01;

import ch.aplu.jgamegrid.*;

/* Klasse Fisch: Superklasse für Spieler und NPCs
Methoden:
act() - Prüft, ob die momentane Richtung erlaubt ist, korrigiert sie falls nötig
und ruft move() auf
move() - Verschiebt den Fisch in x- und y-Richtung gemäss den aktuellen
Geschwindigkeiten
initiate() - Setzt Anfangswerte für Grösse, Geschwindigkeit, Kollision, etc.
Werte aus Fischdaten ein
grow() - Lässt Fisch den nächsten Sprite verwenden, erhöht seine Stufe und passt
Eigenschaften an die neue Stufe an
trytoeat() - frisst einen kleineren* Fisch und ruft danach wenn nötig grow() auf

*Fisch A kann Fisch B nur fressen, wenn die Maulgrösse von A grösser als die
Körpergrösse von B ist und A nicht vergiftet ist
*/
public class Fisch extends Actor{
        public int dx = 0;
        public int dy = 0;
        public int mdx = 6;
        public int mdy = 3;
        public int ddx = 0;
        public int ddy = 0;
        public int bodysize = 1;
        public int mouthsize = 1;
        public int value = 0;
        public int poison = 0;
        public int stage = 0;
        public int eaten = 0;
        public int toeat = 100;
        public int type;
        public int width;
        public int height;
        public int poisoned = 0;
        public boolean movingx = false;
        public boolean movingy = false;
        public boolean alive = true;

       // Wichtiges

         public Fisch(String sprite, int i)
         {
```

```
          super(sprite,10);
        }

      public void move(){
            setLocation(new Location(getLocation().x+dx, getLocation().y
+dy));
        }

        public void act()
        {
            poisoned=Math.max(poisoned-1, 0);
            dx+=ddx;
            dy+=ddy;
            if (ddx < 0){
                setHorzMirror(true);
            } else if (ddx > 0) {
                setHorzMirror(false);
            }
            if (!movingx){
                if (dx>0){
                    dx--;
                } else if (dx<0){
                    dx++;
                }
            }
            if (!movingy){
                if (dy>0){
                    dy--;
                } else if (dy<0){
                    dy++;
                }
            }
            if (Math.abs(dx) > Math.abs(mdx)){
                dx=mdx*(dx/Math.abs(dx));
                ddx=0;
            }
            if (Math.abs(dy) > Math.abs(mdy)){
                dy=mdy*(dy/Math.abs(dy));
                ddy=0;
            }
            if (getX() >= 511 && dx > 0 || getX() <= 0 && dx < 0){
                dx=0;
            }
            if (getY() >= 511 && dy > 0 || getY() <= 0 && dy < 0 ){
                dy=0;
            }
            if (poisoned > 0 && dx != 0){
                dx=dx/Math.abs(dx);
            }
            if (poisoned > 0 && dy != 0){
                dy=dy/Math.abs(dy);
            }
          move();
        }

        public void initiate(){
            mdx=Fischdaten.mdx(type, 0);
            mdy=Fischdaten.mdy(type, 0);
            bodysize = Fischdaten.bodysize(type, 0);
            mouthsize = Fischdaten.mouthsize(type, 0);
            value = Fischdaten.value(type, 0);
```

```
            poison = Fischdaten.poison(type, 0);
            width = Fischdaten.width(type, 0);
            height = Fischdaten.height(type, 0);
            toeat = Fischdaten.toeat(type, 0);
          Fischdaten.setCollision(this);
        }

        // verschiedene Methoden

        public void grow(){
              stage++;
              if (stage<10){
              showNextSprite();
              bodysize = Fischdaten.bodysize(type, stage);
              mouthsize = Fischdaten.mouthsize(type, stage);
              value = Fischdaten.value(type, stage);
              poison = Fischdaten.poison(type, stage);
              width = Fischdaten.width(type, stage);
              height = Fischdaten.height(type, stage);
              toeat = Fischdaten.toeat(type, stage);
              mdx = Fischdaten.mdx(type, stage);
              mdy = Fischdaten.mdy(type, stage);
              Fischdaten.setCollision(this);
              } else {
                    stage--;
                    bodysize += toeat;
                    mouthsize += (int) (toeat/10);
                    toeat = Fischdaten.toeat(type, stage);
              }

        }

        public void trytoeat(Fisch prey){
              if (mouthsize > prey.bodysize && poisoned <= 0){
                    eaten += prey.bodysize;
                    poisoned += prey.poison;
                    prey.alive = false;
                    if (eaten >= toeat){
                          eaten -= toeat;
                          grow();
                    }
              }
        }
}
```

```
package package_01;

import java.awt.event.KeyEvent;

import ch.aplu.jgamegrid.GGKeyListener;
/* Klasse Spieler: Vom Spieler durch die Tastatur gesteuerter Fisch
Methoden:
keyPressed(), keyReleased() - Beschleunigt/bremst Spieler, wenn die Pfeiltasten
gedrückt/losgelassen werden
*/
public class Spieler extends Fisch implements GGKeyListener {

      public Spieler(int pctype){
            super(Fischdaten.sprite(pctype), 10);
          type=pctype;
          initiate();
      }
```

```
      public boolean keyPressed(KeyEvent evt) {
          switch (evt.getKeyCode()){
            case KeyEvent.VK_UP:
              ddy=-1;
              movingy=true;
              break;
            case KeyEvent.VK_RIGHT:
              ddx=1;
              movingx=true;
              break;
            case KeyEvent.VK_LEFT:
              ddx=-1;
              movingx=true;
              break;
            case KeyEvent.VK_DOWN:
              ddy=1;
              movingy=true;
              break;
            case KeyEvent.VK_ESCAPE:
            alive=false;
            break;
          }
          return true;
      }

      public boolean keyReleased(KeyEvent evt) {
          switch (evt.getKeyCode()){
            case KeyEvent.VK_UP:
              ddy=0;
              movingy=false;
              break;
            case KeyEvent.VK_RIGHT:
              ddx=0;
              movingx=false;
              break;
            case KeyEvent.VK_LEFT:
              ddx=0;
              movingx=false;
              break;
            case KeyEvent.VK_DOWN:
              ddy=0;
                movingy=false;
              break;
          }
    return true;
      }

}
```

```
package package_01;

/* Klasse NPC: Computergesteuerter Fisch
Methoden:
act() - Wie bei Fisch, ruft aber zuerst noch getIdea() auf
spawn() - legt Startwerte für Grösse, Position und Richtung fest
getIdea() - steuert NPC, für verschiedene Fischarten verschiedene
Bewegungsmuster
*/

public class NPC extends Fisch {
      public int t=0;
```

```
	public NPC(int npctype) {
		super(Fischdaten.sprite(npctype), 10);
	    type=npctype;
	    initiate();
	}


	public void act(){
		getIdea();
		poisoned--;
		dx+=ddx;
		dy+=ddy;
		if (ddx < 0){
			setHorzMirror(true);
		} else if (ddx > 0) {
			setHorzMirror(false);
		}
		if (dx > mdx){
			dx=mdx;
			ddx=0;
		} else if (dx < mdx*-1){
			dx=-mdx;
			ddx=0;
		}
		if (dy > mdy){
			dy=mdy;
			ddy=0;
		} else if (dy < mdy*-1){
			dy=-mdy;
			ddy=0;
		}
		if ((getX() >= 511+width && dx > 0 )||( getX() <= -width && dx < 0))
{
			dx=0;
			gameGrid.removeActor(this);
			alive=false;
		}
		if (getY() >= 511+height && dy > 0 || getY() <= -height && dy < 0 ){
			dy=0;
			gameGrid.removeActor(this);
			alive=false;
		}
		move();
		t++;
	}

	public void spawn(){
		initiate();
		int spawnstage = (int) (Math.random()*10);
		if (spawnstage >= 8){spawnstage = (int) (Math.random()*10);}
		if (spawnstage >= 6){spawnstage = (int) (Math.random()*10);}
		if (spawnstage > 4){spawnstage = (int) (Math.random()*10);}
		if (spawnstage > 2){spawnstage = (int) (Math.random()*10);}//billige
Art, um kleinere Fische zu bekommen
		for (int  i=0; i<spawnstage; i++){
			grow();
		}
		switch(type){
		case 1:
		case 3:
```

```
			case 4:
				if(Math.random()<0.5){
					setX(-width);
					dx=1;
				}else{
					setX(511+width);
					dx=-1;
				}
				setY((int) (Math.random()*511));
				break;
			case 5:
			case 6:
				setX((int) (Math.random()*511));
				setY(511+height);
				break;
			}
			t=0;
		}

		public void getIdea(){
			switch (type){
			case -1:
				if (stage==5 || stage==6){
					dx=-mdx;
				}
			case 1:
			case 3:
				if (dx>=0){
					ddx+=1;
				} else {
					ddx-=1;
				}
				break;
			case 4:
				if (dx>=0){
					ddx+=1;
				} else {
					ddx-=1;
				}
				if (t%10 == 0){
					dy+=Math.round(Math.random()*2-1);
				}
				break;
			case 5:
				dy=(int) -(Math.pow(Math.sin(t*Math.PI/15),2)*mdy);
				break;
			case 6:
				dy--;
			}
		}

}
```

```
package package_01;

import java.awt.Point;
/* Klasse Fischdaten: Übergibt auf Befehl Werte für die Eigenschaften von
Fischen einer bestimmten Art und Stufe
Methoden:
setCollision() - Legt direkt die für die Kollision mit anderen Fischen benötigte
Form des Fisches fest
alles andere - Gibt den Standardwert der gewünschten Eigenschaft für die
```

```
gegebene Fischsorte zurück
*/
public abstract class Fischdaten {

	public static String sprite(int type){
		switch(type){
		case -2:
			return "sprites/Levelselect.png";
		case -1:
			return "sprites/Knopffisch.png";
		case 0:
			return "sprites/Rotfisch.png";
		case 1:
			return "sprites/Futfisch.png";
		case 2:
			return "sprites/Gruenfisch.png";
		case 3:
			return "sprites/Futterfisch.png";
		case 4:
			return "sprites/Fressfisch.png";
		case 5:
			return "sprites/Inky.png";
		case 6:
			return "sprites/Qualle.png";
		}
		return "";
	}

	public static int bodysize(int type, int num){
		switch(type){
		case -2://Levelselect(dummy)
			return 0;
		case -1://Knopffisch(dummy)
			return 0;
		case 0://Rotfisch
			return 30*(num+1);
		case 1://Futfisch
			return 28*(num+1);
		case 2://Grünfisch
			int [] gruenfischbodysize =
{452,766,1295,2192,3710,6277,10622,17975,30417,51472};
			return gruenfischbodysize[num];
		case 3://Futterfisch
			int [] futterfischbodysize =
{113,206,377,688,1256,2292,4184,7637,13941,25447};
			return futterfischbodysize[num];
		case 4://Fressfisch
			int [] fressfischbodysize =
{1976,2826,4041,5778,8262,11815,16895,24159,34546,49400};
			return fressfischbodysize[num];
		case 5://Inky
			int [] inkybodysize =
{314,524,874,1458,2432,4058,6768,11290,18833,31416};
			return inkybodysize[num];
		case 6://Qualle
			int [] quallebodysize =
{157,246,385,602,942,1474,2306,3608,5647,8836};
			return quallebodysize[num];
		}
		return 0;
	}
```

```
	public static int mouthsize(int type, int num){
		switch(type){
		case -2:
			return 0;
		case -1:
			return 0;
		case 0:
			return 30*(num+1);
		case 1:
			return 28*(num+1);
		case 2:// 30% bodysize
			return (int) ((double)bodysize(type,num)*0.3);//hässlich, aber
irgendwie hat es erst so funktioniert
		case 3:// 20% bodysize
			return (int) ((double)bodysize(type,num)*0.2);
		case 4:// 40% bodysize
			return (int) ((double)bodysize(type,num)*0.4);
		case 5:// 35%
			return (int) ((double)bodysize(type,num)*0.35);
		case 6:// 70%
			return (int) ((double)bodysize(type,num)*0.7);
		}
		return 0;
	}

	public static int value(int type, int num){
		switch(type){
		case -2:
			return 0;
		case -1:
			return 0;
		case 0:
			return 30*(num+1);
		case 1:
			return 28*(num+1);
		case 2:
			return 10*(num+1);
		case 3:
			return 10*(num+1);
		case 4:
			return 15*(num+1);
		case 5:
			return 30*(num+1);
		case 6:
			return 4*(num+1);
		}
		return 0;
	}

	public static int poison(int type, int num){
		switch(type){
		case -2:
			return 0;
		case -1:
			return 0;
		case 0:
			return 0;
		case 1:
			return 0;
		case 2:
```

```
                return 0;
            case 3:
                return 0;
            case 4:
                return 0;
            case 5:
                return 0;
            case 6:
                return 10+num*num;
            }
            return 0;
        }

        public static int width(int type, int num){
            switch(type){
            case -2:
                return 88;
            case -1:
                return 90;
            case 0:
                return 51*(num+1);
            case 1:
                return 50*(num+1);
            case 2:
                int [] gruenfischwidth = {24,32,40,52,68,90,116,152,196,256};
                return gruenfischwidth[num];
            case 3:
                int [] futterfischwidth = {12,16,22,30,40,54,73,99,133,180};
                return futterfischwidth[num];
            case 4:
                int [] fressfischwidth =
{52,62,74,88,106,127,152,182,217,260};
                return fressfischwidth[num];
            case 5:
                int [] inkywidth = {20,26,33,43,56,72,93,120,155,200};
                return inkywidth[num];
            case 6:
                int [] quallewidth = {20,25,31,39,49,61,77,96,120,150};
                return quallewidth[num];
            }
            return 0;
        }

        public static int height(int type, int num){
            switch(type){
            case -2:
                return 88;
            case -1:
                return 50;
            case 0:
                return 25*(num+1);
            case 1:
                return 25*(num+1);
            case 2:
                int [] gruenfischheight = {24,32,40,52,68,90,116,152,196,256};
                return gruenfischheight[num];
            case 3:
                int [] futterfischheight = {12,16,22,30,40,54,73,99,133,180};
                return futterfischheight[num];
            case 4:
                int [] fressfischheight = {38,45,54,65,78,93,111,133,159,190};
```

```
			return fressfischheight[num];
		case 5:
			int [] inkyheight = {20,26,33,43,56,72,93,120,155,200};
			return inkyheight[num];
		case 6:
			int [] qualleheight = {20,25,31,39,49,61,77,96,120,150};
			return qualleheight[num];
		}
		return 0;
	}

	public static int mdx(int type, int num){
		switch(type){
		case -2:
			return 0;
		case -1:
			if (num==5 || num==6){
				return 7;
			}
			return 0;
		case 0:
			return 10;
		case 1:
			return 6;
		case 2:
			return 11;
		case 3:
			return 6;
		case 4:
			return 7;
		case 5:
			return 3;
		case 6:
			return 2;
		}
		return 0;
	}

	public static int mdy(int type, int num){
		switch(type){
		case -2:
			return 0;
		case -1:
			return 0;
		case 0:
			return 5;
		case 1:
			return 3;
		case 2:
			return 7;
		case 3:
			return 4;
		case 4:
			return 4;
		case 5:
			return 10+2*num;
		case 6:
			return 4;
		}
		return 0;
	}
```

```
	public static int toeat(int type, int num){
		switch(type){
		case -2:
			return 100;
		case -1:
			return 100;
		case 0:
			int [] rotfischtoeat =
{40,90,150,220,300,390,490,600,720,10000};
			return rotfischtoeat[num];
		case 1:
			int [] futfischtoeat =
{40,90,150,220,300,390,490,600,720,10000};
			return futfischtoeat[num];
		case 2://bodysize(stage+1)
			int [] gruenfischtoeat =
{766,1295,2192,3710,6277,10622,17975,30417,51472,87102};
			return gruenfischtoeat[num];
		case 3:
			int [] futterfischtoeat =
{206,377,688,1256,2292,4184,7637,13941,25447,46449};
			return futterfischtoeat[num];
		case 4:
			int [] fressfischtoeat =
{2826,4041,5778,8262,11815,16895,24159,34546,49400,70641};
			return fressfischtoeat[num];
		case 5:
			int [] inkytoeat =
{524,874,1458,2432,4058,6768,11290,18833,31416,52406};
			return inkytoeat[num];
		case 6:
			int [] qualletoeat =
{246,385,602,942,1474,2306,3608,5647,8836,13826};
			return qualletoeat[num];
		}
		return 0;
	}

	public static void setCollision(Fisch fish){
		switch(fish.type){
		case -2:
		case -1:
		case 0:
		case 1:
		case 2:
		case 3:
		case 5:
		case 6:
		    fish.setCollisionCircle(fish.stage, new Point(0,0),
fish.height/2);
		    break;
		case 4:
			fish.setCollisionRectangle(fish.stage, new Point(0,0),
fish.width, fish.height);
			break;
		}
	}
}
```

```
package package_01;
```

```
import java.awt.Color;
import java.awt.event.KeyEvent;

import ch.aplu.jgamegrid.*;
/* Klasse BLInput: Eingabefeld für die Bestenliste
Methoden:
keyPressed() - Fügt den eingegebenen Buchstaben (falls erlaubt) dem angezeigten
String hinzu, beendet sich bei Enter und löscht den letzten Buchstaben bei
Delete/Backspace
*/

public class BLInput extends GGTextField implements GGKeyListener{
	String drin = "Fritzi";
	String allowedchars =
"abcdefghijklmnopqrstuvwxyzQWERTZUIOPASDFGHJKLYXCVBNMäöüÄÖÜàéè1234567890+*ç%&/
()=.,-_+\\<> ";
	public boolean finished = false;

	public BLInput(GameGrid gg, String text, Location location, boolean
enableRefresh) {
		super(gg, text, location, enableRefresh);

		this.setBgColor(new Color(63,127,255));
		this.setTextColor(Color.BLACK);
		this.show();
	}

	public boolean keyPressed(KeyEvent evt) {
		if (allowedchars.contains(""+evt.getKeyChar())){
			drin=drin+evt.getKeyChar();
			this.setText(drin);
		} else if(evt.getKeyCode()==KeyEvent.VK_ENTER){
			finished=true;
		} else if ((evt.getKeyCode()==KeyEvent.VK_DELETE ||
evt.getKeyCode()==KeyEvent.VK_BACK_SPACE) && drin.length()>0){
			drin=drin.substring(0, drin.length()-1);
			this.setText(drin);
		}
		return false;
	}

	public boolean keyReleased(KeyEvent evt) {
		return false;
	}

}
```

7. Resultate und Testen

Testen: Das Testen des Spiels verlief hauptsächlich ohne Probleme. Die einzige grössere Schwierigkeit war, dass die Gamegrid-Klasse Actor für jeden Sprite eine eigene Form für die Kollisionserkennung hat. Da die Fische beim Wachsen ihren Sprite ändern musste dies berücksichtigt werden.

Resultate: Beim Spielstart befindet man sich im Menü (sichtbar auf Abbildung 5. Um zu spielen, die Bestenliste anzusehen oder das Spiel zu schliessen, muss man den entsprechend angeschriebenen Fisch fressen. Die Steuerung durch den Spieler erfolgt durch die Pfeiltasten und während des Spiels kann man über die Escape-Taste jederzeit aufhören. Die errungenen Punkte werden oben links angegeben, während der Balken im unteren Bereich des Fensters anzeigt, wie viel der Spieler noch fressen muss, bis er sich das nächste Mal vergrössert (Abbildung 6). Sollte der Spieler am Ende des Spiels eine hohe Punktzahl erreicht haben, wird er aufgefordert, seinen Namen in die Bestenliste einzugeben (Abbildung 7). Im einen Level gibt es verschiedene gegnerische Fischarten, darunter den Fressfisch (in der Abbildung 8 ist ein kleiner Fressfisch oben links und ein grösserer zum Teil im oberen rechten Bereich des Spielfelds), der seine Richtung ändern kann, und die Qualle, die den Fisch, der sie auffrisst, vergiftet, wodurch dieser langsam wird und nichts fressen kann. Wenn der Spieler vergiftet ist, merkt man dies daran, dass Punkteanzeige und Fressbalken grün gefärbt sind (auch sichtbar auf Abbildung 8).

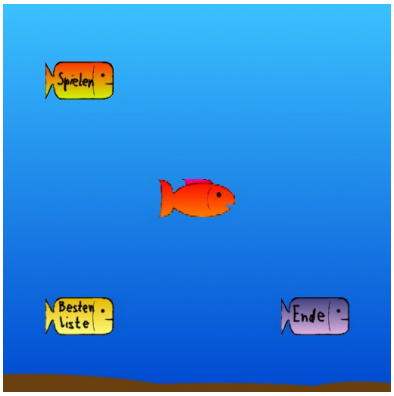

Abbildung 5: Spielmenü

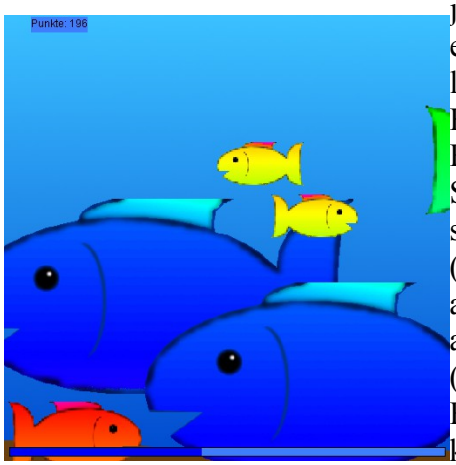

Abbildung 6: Game Over steht bevor

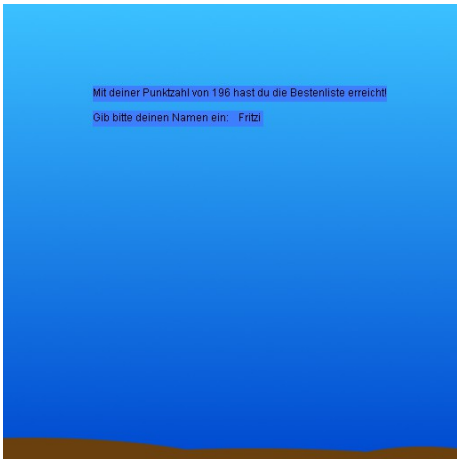

Abbildung 7: Glückwunsch!

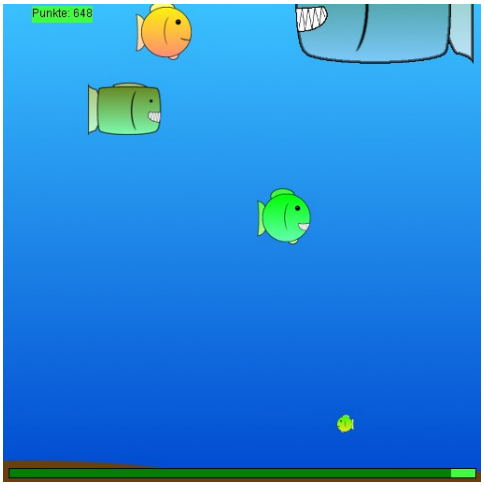

Abbildung 8: Spiel als Grünfisch

8. Diskussion und Ausblick

Diskussion: Bei der Arbeit am Fischgame habe ich Verschiedenes gelernt, darunter der Umgang mit Gamegrid, Verwendung von switch in Java und wie man aus Dateien liest, bzw. in sie schreibt.
Die festgelegten Ziele wurden erreicht, wobei das Kriterium „Mehrere Levels“ nur knapp erfüllt wurde. Es gibt in der Version 1.0 zwei Levels, so dass man gerade mal den Plural verwenden kann.

Dafür hat es im einen davon drei Fischarten mit besonderen Eigenschaften.
Bei der Programmierung ergaben sich vor allem zwei grosse Probleme: Zuerst wurde klar, dass man Bilder nicht einfach skalieren konnte, so dass die Fische für jede Grösse ein eigenes Bild benötigten. Das andere Problem war dann, dass jedes Bild seine eigenen Kollisionsdaten benötigte, was nach langem Probieren endlich herausgefunden und dann schnell gelöst wurde.

Ausblick: Das Hinzufügen von neuen Levels wurde etwas aufgeschoben, da es dabei vor allem darum geht, neue Fischarten zu zeichnen und ihre Daten einzugeben, was relativ Zeitaufwendig ist. Programmieren muss man dazu kaum, da der Code so konzipiert wurde, dass man sehr leicht in der Klasse Fischdaten neue Fischarten erstellen kann. Einige andere Erweiterungen, die noch gemacht werden können wären zum Beispiel getrennte Bestenliste für die verschiedenen Levels, die Möglichkeit, das Spiel zu pausieren und vielleicht ein Sandbox-Modus, wo beliebige Fischarten gespielt werden können.