Christoph-Scheiner-Gymnasium Ingolstadt

Seminararbeit
aus dem
wissenschaftspropädeutischen
Seminar
im Fach Informatik

Implementierung des Spiels „Reverse PacMan“ als Browser-Game

Angefertigt von	Benjamin Kass
Reifeprüfungsjahrgang	2022
Kursleiter	StD Martin Pabst

Inhaltsverzeichnis

1 Einleitung: 3

1.1 Spielidee 3

1.2 Spielprinzip 3

1.3 Spielaufbau 4

2 Erschaffung einer virtuellen Welt 5

2.1 Das Labyrinth 5

2.1.1 Das Labyrinth im Code 5

2.2 Die Giftwolken 6

3 Der Algorithmus 7

3.1 Wie bleibt PacMan weg von den Geistern 7

3.2 Der Algorithmus im Code 8

3.2.1 Der Algorithmus der Geister 8

3.2.2 Der Algorithmus bei PacMan 9

3.3 Wie bewegt sich PacMan zu den Punkten 11

4 Das Movement der Geister 13

5 Mögliche Erweiterungen/Verbesserungen 16

5.1 Die Spielschwierigkeit 16

5.2 Die Punkte 16

5.3 Lokaler Multiplayer 17

6 Fazit/Schlusswort 18

7 Literaturverzeichnis 19

8 Eidesstattliche Erklärung 20

1 Einleitung:

1.1 Spielidee

Bevor man ein Spiel programmieren kann, braucht man eine Idee. Diese muss jedoch auch Rahmenbedingungen entsprechen. Das Spiel darf nicht zu umfangreich sein, sodass man es neben der Schule ohne zu viel Aufwand programmieren kann. Außerdem sollte es nicht zu anspruchsvoll oder kompliziert sein. Ich habe mich deshalb an simplen, älteren Arcade-Games orientiert. Allerdings wollte ich nicht einfach nur ein bereits existierendes Spiel nachprogrammieren, sondern durch meine eigenen Ideen das Spiel „neu“ gestalten. So entstand die Spielidee von „Reverse PacMan“. Ein klassisches Arcade-Game, aber abgeändert.

1.2 Spielprinzip

In Reverse PacMan sammelt nicht der Spieler mit PacMan Punkte, sondern mit den Geistern PacMan. Ziel des Spiels ist es nämlich, mit den vier Geistern PacMan zu fangen, bevor dieser alle Punkte gesammelt hat, um ins nächste Level zu kommen. Doch nicht nur das. Es erscheinen auch zufällig kleine Giftwolken irgendwo auf dem Spielfeld. Befindet sich ein Geist in der Giftwolke, löst er sich auf und verschwindet. Zumindest für das aktuelle Level. Schafft man ein Level, wird ein Geist wiederbelebt, sofern man einen verloren hat. Verliert man also zwei von vier Geistern und fängt PacMan trotzdem, startet man im nächsten Level mit drei Geistern. Auch wird pro Level die Spielgeschwindigkeit erhöht, allerdings nur zu einer Wahrscheinlichkeit von 10%. Damit das Spiel spielbar bleibt, es gibt ein Maximum für die Geschwindigkeit. Sollten alle vier Geister von den Giftwolken getroffen werden oder PacMan alle Punkte gesammelt haben, ist das Spiel zu Ende. Beim nächsten Mal startet man wieder in Level eins mit allen vier Geistern bei normaler Geschwindigkeit und hat erneut die Chance, seinen Highscore zu knacken.

1.3 Spielaufbau

Reverse PacMan besteht aus drei verschiedenen Szenen. Ein Startbildschirm, ein Game-Over-Screen und dem eigentlichen Spiel. Der Startbildschirm besteht aus dem Logo des Spiels, dem Startknopf „Play“, um das Spiel zu starten sowie einer kleinen Animation, in der PacMan von einem Geist gejagt wird. Der Game-Over-Screen ist ähnlich aufgebaut, nur statt dem Logo wird „Game Over“ angezeigt und darunter ist ein „Retry”-Button.

In der Spielszene sieht man das Labyrinth, in dem sich in den Ecken die vier Geister und im Zentrum Pacman befinden. Oben in der Mitte wird das aktuelle Level angezeigt, links oben der Highscore. In der rechten oberen Ecke, befindet sich außerdem ein „Mute-Button”, welcher die Audio von PacMan stummschaltet. Startet man das Spiel (neu), blinken im Vordergrund die Pfeiltasten, welche als Steuerungstutorial dienen. Der Hintergrund wird dabei leicht verdunkelt. Drückt man nun eine der vier Pfeiltasten, beginnt das Spiel. Das Tutorial gibt es nur in Level 1, doch jedes Level startet erst, sobald der Spieler sich bewegt.

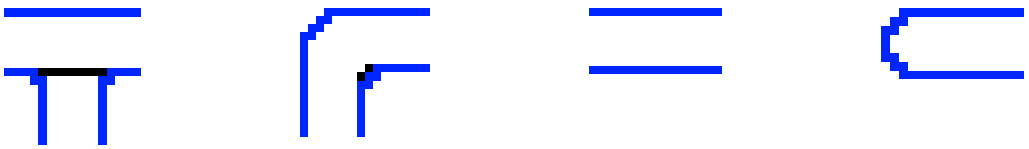
2 Erschaffung einer virtuellen Welt

2.1 Das Labyrinth

Das wichtigste der graphischen Gestaltung des Spiels ist das Spielfeld. Auch hier habe ich mich am originalen PacMan orientiert. Nur die Struktur und der Aufbau des Labyrinths sind unterschiedlich.

2.1.1 Das Labyrinth im Code

Da ich mich dem Spielfeld sehr früh gewidmet habe, wusste ich zu der Zeit noch nicht, welche die besten Herangehensweisen für die Kollisionen und Steuerungen der Geister sind. Mein Plan damals war es, alles mit Physik zu lösen. Somit brauchte das Labyrinth einen „Body“, durch den die Geister nicht hindurch kommen. Weil ich hierzu nichts im Internet gefunden habe und selbst auch nicht weiter wusste, wie man es schöner lösen kann, habe ich mich dazu entschieden, das Spielfeld aus einzelnen „Puzzleteilen“, die jeweils rechteckige Bodys haben, zusammenzusetzen:[1]



Das war natürlich sehr umständlich. Heute ist das Labyrinth eine einzige, ganze PNG-Datei. Ermöglicht wurde dies durch einen zweidimensionalen Array, der das Spielfeld für alle Spielfiguren repräsentiert. Der Array liegt genau passend über dem Labyrinth und regelt das Movement und Kollisionen der Geister und von PacMan. Auch wird der Algorithmus, dass PacMan den Geistern fern bleibt und die Punkte sammelt, deutlich einfacher.

Jedes Feld in dem Array bekommt einen Wert zugewiesen. Die Felder, die eine Wand repräsentieren, bekommen jeweils den Wert ‚–1‘ zugewiesen, alle anderen erstmal ‚0‘.

[1] Die schwarzen Stellen in den Grafiken sind dafür da, mögliche Pixelfehler zu beheben, da oft zwei Puzzleteile übereinander lagen.

2.2 Die Giftwolken

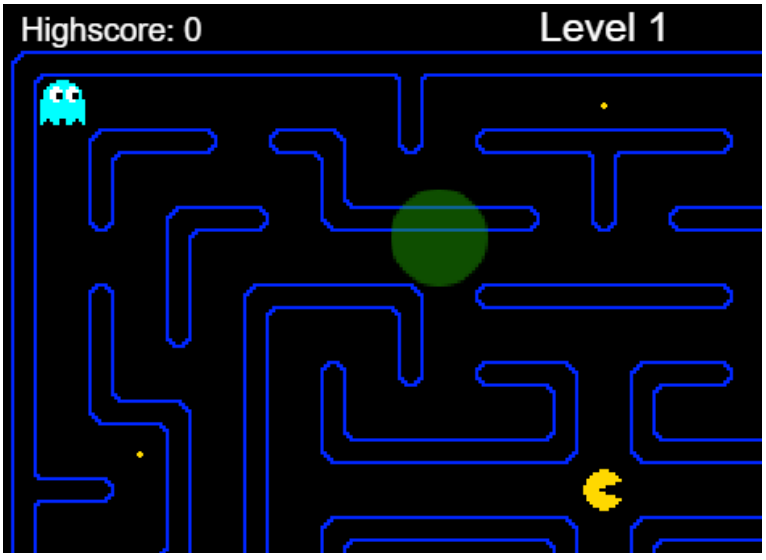
Die Giftwolken sind kleine grüne, leicht durchsichtige Kreise. Sie erscheinen erst wenn das Spiel startet, also der Spieler sich bewegt hat. Mit einer Wahrscheinlichkeit von 0,5% erscheint eine Giftwolke beim Aufruf der Update-Methode, solange die Giftwolke noch nicht aktiv ist.

Damit die Giftwolke im Spielfeld erscheint und nicht über den Rand ragt, werden die x und y Koordinaten eingeschränkt:

```
this.x = Math.floor(Math.random()*650) + 60;

this.y = Math.floor(Math.random()*450) + 60;
```

Nach fünf Sekunden wird die Giftwolke wieder unsichtbar und auf die Koordinaten (0,0) gesetzt, da sie trotzdem immer noch aktiv ist, was sie eigentlich nicht sein sollte.

```
this.time.delayedCall(5000, () => {
      this.setVisible(false);
            this.setActive(false);
            this.x = 0;
            this.y = 0;
            this.poisonactive = false;
})
```

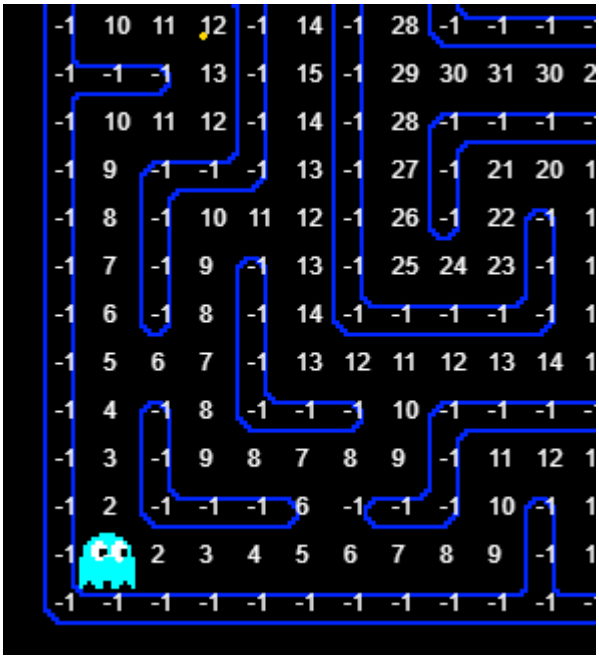


3 Der Algorithmus

3.1 Wie bleibt PacMan weg von den Geistern

Dadurch, dass ein zweidimensionaler Array das Spielfeld repräsentiert kann jedem Feld in dem Array ein Wert zugewiesen werden. Die Idee ist, den Array mit Werten zu füllen und je höher dieser Wert ist, desto besser ist das Feld für PacMan. Die einzelnen Felder werden von den Geistern aus aufsteigend gefüllt. Das Feld auf dem sich ein Geist befindet, bekommt auch den Wert ‚–1', da die Geister nicht durch andere Geister laufen sollen. Dadurch wird verhindert, dass alle Geister zu „einem Geist" werden.

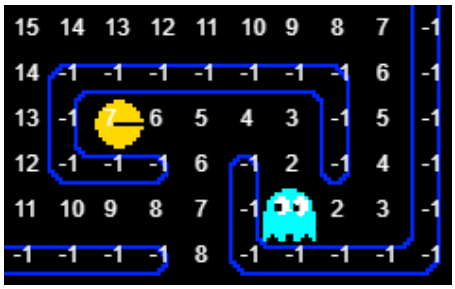
Zur Veranschaulichung steht in der folgenden Abbildung jede Zahl für den Wert, der in dem entsprechenden Feld ist:



PacMan muss nun nur die vier Felder, die um ihn herumliegen, prüfen und das Feld mit dem größten Wert finden. So bewegt er sich automatisch weg von den Geistern.

Das ist schon eine ganz gute Lösung, doch PacMan bleibt oft in Sackgassen stecken und er sucht sich nicht den besten Weg den Geistern fernzubleiben.

Da er nur die Felder um sich herum prüft, sieht er hier drei Wände und ein Feld mit einem kleineren Wert, wenn er es mit seinem aktuellen vergleicht. Also bleibt PacMan in der Sackgasse, obwohl ein paar Felder weiter, viel höhere Werte liegen.



Eine bessere Lösung wäre es, wenn PacMan mehrere Felder überprüft, um nach einem möglichen, größeren Wert zu suchen, welcher „hinter“ niedrigeren Werten liegt. Dann erstellt er einen weiteren zweidimensionalen Array, welcher von dem gefundenen Maximum, aufsteigend mit Zahlen gefüllt wird. Überprüft PacMan nun die vier Felder in dem neuen Array um ihn herum, muss er immer das niedrigste nehmen und wird so automatisch zu dem besseren Feld geführt. PacMan sucht mit einer Reichweite von 15 Feldern entlang seiner möglichen Laufwege nach einem Maximum, um es nicht unmöglich für den Spieler zu machen, da PacMan nun perfekt ausweichen kann und es extrem schwer geworden ist, ihn zu fangen.

3.2 Der Algorithmus im Code

3.2.1 Der Algorithmus der Geister

Der Algorithmus wird rekursiv[2] für jedes Feld aufgerufen. Angefangen bei der Klasse Reverse_PacMan (Die Hauptszene). In der Update Methode wird, in Abhängigkeit der Spielgeschwindigkeit, das `gameField[x][y]` geleert.

Dann wird der Startmethode für den Algorithmus bei den Geistern das `gameField[x][y]` übergeben. Auch wird die Methode `considerMove()` bei PacMan aufgerufen, welche PacMan bewegen lässt.

Beim Geist startet nun die rekursive Methode. Zuerst wird dem aktuellen Feld der aktuelle `value`, startend bei ‚1‘ zugewiesen. Dann wird die Zuweisung wellenartig vom Geist aus durchgeführt. Ist der Wert des Feldes schon ‚–1‘, darf er nicht überschrieben werden, da sonst eine Wand „zerstört“ wird.

[2] rekursiv: Eine Methode ruft sich selbst erneut auf

Die Methode ruft sich nun selbst erneut auf, diesmal mit einem um ‚1' erhöhtem `value` und veränderter x bzw. y Koordinate, je nachdem in welche Richtung der Algorithmus weiterläuft. Der Algorithmus stoppt, wenn er auf ein Feld trifft, das einen kleineren Wert hat, jedoch nicht ‚0' ist, da das leere `gameField[x][y]` mit Nullen gefüllt ist[3]. Ist der Wert nämlich kleiner, befindet sich ein anderer Geist näher an diesem Feld und somit wäre der höhere Wert hier falsch.

```
setGameFieldValuesRecursive(gamefield: number[][], x: number, y: number, value: number)
{
        gamefield[x][y] = value;
        value++;
       //Links
       if(x != -1 &&  (gamefield[x-1][y] == 0 || gamefield[x-1][y] > value)){
           this.setGameFieldValuesRecursive(gamefield, x-1, y, value);
        }
       //Rechts
       if(x != -1 && (gamefield[x+1][y] == 0|| gamefield[x+1][y] > value)){
           this.setGameFieldValuesRecursive(gamefield, x+1, y, value);
        }
       //Unten
       if(y != -1 && (gamefield[x][y-1] == 0 || gamefield[x][y-1] > value)){
           this.setGameFieldValuesRecursive(gamefield, x, y-1, value);
        }
       //Oben
       if(y != -1 && (gamefield[x][y+1] == 0 || gamefield[x][y+1] > value)){
           this.setGameFieldValuesRecursive(gamefield, x, y+1, value);
       }
 }
```

3.2.2 Der Algorithmus bei PacMan

Zuerst muss PacMan das Feld mit dem höchsten Wert in seinem Umkreis finden. Dazu wird ein ähnliches Verfahren, wie bei den Geistern benutzt. Die Methode ruft sich rekursiv auf, bis sie auf eine Wand trifft oder die Reichweite überschritten wird. Der höchste Wert wird dann mit seiner x und y Koordinate gespeichert. Wird auf dem Weg ein größerer Wert gefunden wird der gespeicherte Wert einfach überschrieben:

[3] Der Algorithmus würde sonst schon am Anfang abbrechen.

```
findBestFieldRecursive(gamefield: number[][], x: number, y: number, n: number){
        if(gamefield[x][y] > this.bestFieldValue){
            this.bestFieldValue = gamefield[x][y];
            this.bestFieldx = x;
            this.bestFieldy = y;
        }
        //Links
        if(gamefield[x-1][y] != -1 && n > 0){
            this.findBestFieldRecursive(gamefield, x-1, y, n-1);
        }
        //Rechts
        if(gamefield[x+1][y] != -1 && n > 0){
            this.findBestFieldRecursive(gamefield, x+1, y, n-1);
        }
        //Unten
        if(gamefield[x][y-1] != -1 && n > 0){
            this.findBestFieldRecursive(gamefield, x, y-1, n-1);
        }
        //Oben
        if(gamefield[x][y+1] != -1 && n > 0){
            this.findBestFieldRecursive(gamefield, x, y+1, n-1);
        }
}
```

(n ist hier die Reichweite, startend bei 15)

Jetzt muss nur noch das zweite Array mit den passenden Zahlen gefüllt werden, ausgehend von `bestFieldx` und `bestFieldy`, den Koordinaten des Feldes mit dem höchsten Wert in Reichweite. Das Array ist einfach eine Kopie des leeren `gameField[x][y]`, d.h. die Wände sind mit dem Wert ‚–1' eingetragen. Auch hier wird dasselbe Verfahren wie bei den Geistern angewendet. Die rekursive Methode stoppt, wenn sie auf eine Wand trifft oder wenn der Wert vor ihr kleiner ist als der aktuelle und ruft sich selbst mit um ‚1' erhöhter `value` auf. Der Array hat nun den Wert ‚1', bei genau dem Feld, welches PacMan als das für sich am besten geeignetsten gefunden hat.

3.3 Wie bewegt sich PacMan zu den Punkten

Anfangs habe ich geplant, dass PacMan sich gezielt zu den Punkten bewegt. Jetzt ist es etwas abhängiger vom Zufall, weil sich die Werte im `gameField[x][y]` nur um den Punkt herum erhöhen. Jedoch kommen auch hier einige Probleme auf. Addiert man etwa konstant eine höhere Zahl zu den Feldern im Umkreis, kann es sein, dass der Punkt nicht den höchsten Wert bekommt. Addiert man eine höhere Zahl, die sich pro Feld um ‚1' erniedrigt, gleicht sich die Zahl mit den Werten vom `gameField[x][y]` aus und ergeben mehrmals dieselben Werte nebeneinander, was auch nicht praktisch ist, da der Punkt dann wieder nicht den höchsten Wert bekommt. Die beste Möglichkeit ist es, eine höhere Zahl vom Punkt aus zu addieren und immer zwei pro Feld zu subtrahieren. So gleichen sich die Zahlen nicht aus und das Feld mit dem Punkt hat den höchsten Wert. Genau wie bei den Geistern wird hier eine rekursive Methode verwendet, um die Werte zu setzen. Damit nicht dasselbe Feld nochmal besucht wird, speichert ein zweidimensionales Boolean-Array, welche Felder schon besucht worden sind.

```
createVisitedField(){
        this.gameFieldVisited = [];
        for(let i = 0; i < this.emptyGamefield.length; i++){
            this.gameFieldVisited[i] = [];
            for(let j = 0; j < this.emptyGamefield[i].length; j++){
                this.gameFieldVisited[i][j] = false;
            }
        }
}
```

Kommt PacMan in die Nähe eines Punktes, erkennt er die hohen Werte beim Punkt und setzt diese automatisch als sein Ziel, da sie größer als die anderen, durchschnittlichen Werte sind. Da der größte Wert genau auf dem Punkt liegt, wird PacMan genau zu dem Punkt geleitet.

Hier sieht man ein Beispiel:

Der rote Kreis markiert die Position des Punktes.

Alle Felder auf die der Punkt einen Einfluss hat, sind hier gelb gefärbt.

Kommt PacMan nun mit seiner Reichweite auf ein gelb markiertes Feld, geht er automatisch zum Punkt.

Ein Nachteil davon ist, dass sich ein Geist auf bzw. „hinter” einem Punkt verstecken kann, um dann auf PacMan zu warten, da der Punkt die niedrigen Felder, die der Geist setzt, so erhöht, dass PacMan trotzdem noch zum Punkt will.

Man kann es aber auch als taktischen Spielzug sehen, welchen die Spieler herausfinden können, um PacMan zu fangen, denn das Spiel ist schon schwer genug.

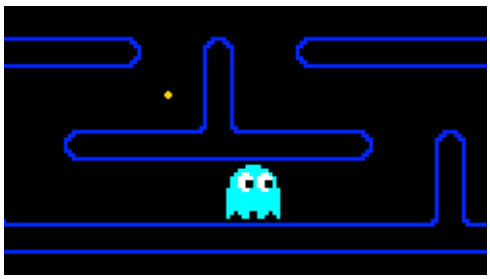
4 Das Movement der Geister

Die Steuerung und Bewegung der Geister wollte ich auch dem originalen PacMan nachahmen. Hierbei werden die Geister mit den vier Pfeiltasten gesteuert und sollen so lange in eine Richtung laufen, bis die Richtung durch den Spieler geändert wird oder sie auf eine Wand treffen. Für die Umsetzung wird die aktuelle Richtung in booleschen Variablen gespeichert:

```
left:boolean = false;

right:boolean = false;

up:boolean = false;

down:boolean = false;
```

Je nachdem welche Pfeiltaste gedrückt wird, wird die jeweilige Variable auf `true` gesetzt, die anderen dementsprechend wieder auf `false`. Eine Sache gibt es jedoch noch zu beachten:

Dieser Geist bewegt sich im Moment nach rechts. Wenn der Spieler jetzt die Pfeiltaste nach oben drückt, soll der Geist trotzdem weiter nach rechts laufen und nicht nach oben schauen und gegen die Wand laufen.



Das bedeutet, dass bevor die jeweilige Variable überhaupt auf `true` gesetzt wird, muss überprüft werden, ob das `gameField[x][y]` an dieser Stelle den Wert ‚–1' besitzt.

```
if(this.cursors.left.isDown && this.gamefield[this.fieldcoordx - 1][this.fieldcoordy] != -1){

        this.left = true;

        this.right = false;

        this.down = false;

        this.up = false;

}
```

(hier: nur das Beispiel für die linke Pfeiltaste)

Der nächste Schritt ist den Geist bewegen zu lassen. Die Schwierigkeit hierbei ist, dass der Geist sich im Raster bzw. `gameField[x][y]` bewegen soll[4], seine Geschwindigkeit erhöhen kann und er eine “weiche” Bewegung hat, bei der er sich nicht kleine Abschnitte teleportiert, d.h. er soll sich konstant von Pixel zu Pixel bewegen, ohne welche zu überspringen.

Zuerst wird die Textur des Geistes geändert. Es ändert sich aber nur die Position der Augen, die in die vier verschiedenen Richtungen schauen:

Als nächstes wird der Wert im `gameField[x][y]` auf ‚0‘ gesetzt, da dieser im Moment ‚–1‘ beträgt und somit nicht überschrieben werden kann, was aber falsch ist, da sich der Geist dann nicht mehr auf diesem Feld befindet.

Danach wird die entsprechende Feldkoordinate `fieldcoordx` oder `fieldcoordy` des Geistes um ‚1‘/‚–1‘ geändert, je nachdem in welche Richtung sich der Geist bewegt.

Jetzt werden die Variablen `destX` bzw. `destY` um ‚24‘/‚–24‘ geändert. Sie speichern die Zielkoordinaten des Geistes.

```
if(this.left){
        this.setTexture("ghostleft");
        this.gamefield[this.fieldcoordx][this.fieldcoordy] = 0;
        this.fieldcoordx -= 1;
        this.destX -= 24;
}
```

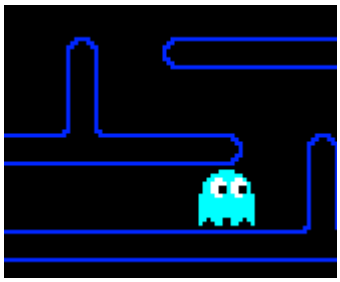
(hier: nur das Beispiel für die linke Richtung)

[4] Der Abstand zwischen zwei Feldern im `gameField[x][y]` beträgt genau 24 Pixel.

Wurden nun alle Richtungen durchlaufen, wird die Distanz zwischen den aktuellen Koordinaten und den Zielkoordinaten berechnet und bei Bedarf geändert:

```
//Für x
let distX = Math.abs(this.x - this.destX);
 if(distX > 0.0001 ){
     this.x += Math.sign(this.destX - this.x)*Math.min(distX, this.speed);
 }
 //Für y
 let distY = Math.abs(this.y - this.destY);
 if(distY > 0.0001 ){
     this.y += Math.sign(this.destY - this.y)*Math.min(distY, this.speed);
 }
```

Durch dieses Verfahren wird es auch ermöglicht, schon bevor der Geist eine Kurve erreicht, die entsprechende Pfeiltaste zu drücken und der Geist wird bei der Kurve in diese Richtung abbiegen:



Wird jetzt die Pfeiltaste nach oben gedrückt, wird der Geist, wenn er die Wand rechts erreicht hat, nach oben abbiegen, obwohl im Moment über ihm eine Wand ist.

5 Mögliche Erweiterungen/Verbesserungen

5.1 Die Spielschwierigkeit

Das erste große Problem ist die „Balance” des Spiels, also wie schwer oder leicht es zu gewinnen ist. Das hat sich jedoch während des Programmierens immer wieder stark geändert. Im Moment ist es so, dass wenn man einige Zeit gespielt hat, einige Taktiken oder Verhaltensmuster von PacMan kennt und es somit sehr einfach ist, oft und schnell zu gewinnen. Auf der anderen Seite ist es für jemanden, der das Spiel zum ersten Mal spielt, extrem schwer, da PacMan nun ziemlich gut ausweichen kann. Durch die beschränkte Reichweite und die hohe Priorisierung der Punkte habe ich versucht ein gutes Mittelmaß zu finden.

Das Problem könnte man lösen, indem man drei Schwierigkeitsstufen einbaut. Je nach Schwierigkeitsstufe hat dann PacMan eine höhere Reichweite. Bei dem leichtesten Modus könnte man PacMan sogar etwas langsamer machen, sodass man ihn durch stumpfes verfolgen nach einiger Zeit auch fangen könnte.

5.2 Die Punkte

Nur vier Punkte im ganzen Spielfeld zu haben sieht etwas komisch aus, da man es von PacMan anders gewohnt ist. Die eigentliche Idee warum ich nur mit vier Punkten angefangen habe war, dass ich wusste, das Spiel könnte schwer werden, wenn PacMan einen vernünftigen Algorithmus bekommt. Damit lag ich zwar mit meiner Vermutung richtig, aber es wurde dadurch auch nicht gerade leichter. Durch die geringe Anzahl an Punkten hat man nur sehr wenig Zeit PacMan zu fangen und man darf die Punkte nie ganz alleine lassen.

Ich könnte mir vorstellen, wenn das Spiel auf jedem Feld einen Punkt hat, dass es dann schöner und einfacher ist zu spielen. Allerdings weiß ich auch nicht, wie genau PacMan sich dann verhalten wird und ob er so überhaupt noch den Geistern ausweichen kann, da die Punkte überall die Werte erhöhen.

5.3 Lokaler Multiplayer

Natürlich macht ein Spiel zu zweit mehr Spaß als alleine. Also hab ich mir Gedanken dazu gemacht, wie man Reverse PacMan zu zweit spielen könnte.

Meine Idee ist, dass zwei Geister von den Tasten ‚W-A-S-D' und zwei Geister von den Pfeiltasten gesteuert werden. Das ist die simpelste Lösung für einen Multiplayer.

Ob alleine oder zu zweit gespielt wird, kann dann über weitere Buttons im Hauptmenü entschieden werden.

6 Fazit/Schlusswort

Reverse PacMan war eine Idee, bei der ich anfangs nicht wusste, wie ich es umsetzten könnte oder ob ich es überhaupt umsetzen kann. Deshalb stellte das Projekt auch eine große Herausforderung für mich da. Die Ideen kamen erst nach und nach, was man auch daran erkennen kann, dass ich manche Sachen später anders gemacht habe, als sie eigentlich gedacht waren.

Mit dem Endergebnis meines Projektes bin ich trotzdem sehr zufrieden. Ich hab mehr erreicht als ich erwartet habe und freue mich jedes Mal, wenn ich sehe, wie gut PacMan von den Geistern wegläuft. Natürlich kann man das Spiel noch erweitern. Dafür habe ich auch schon ein paar Ideen, doch mein Spiel ist zu dem jetzigen Zeitpunkt eine gute Grundlage, bei der das Spielen Spaß macht.

Die wichtigste Erkenntnis die ich für mich aus dem Projekt mitnehme ist, dass man immer einen genauen Plan haben sollte, bevor man zu Programmieren beginnt. Das erspart viel Arbeit und Zeit, was im späteren Verlauf vieles erleichtert.

Abschließend kann ich sagen, dass mir das Arbeiten an dem Projekt sehr viel Spaß gemacht hat und ich dabei auch sehr viel Neues lernen konnte, da es eine neue Umgebung und Programmiersprache für mich war.

7 Literaturverzeichnis

1.1 Typescript - multidimensional array initialization

https://stackoverflow.com/questions/30144580/typescript-multidimensional-array-initialization

aufgerufen am 13.07.2021

1.2 Phaser 3 API Documentation - Class: Sprite

https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Sprite.html

aufgerufen am 23.08.2021

1.3 Phaser - Examples - Rectangle

https://phaser.io/examples/v3/view/game-objects/shapes/rectangle

aufgerufen am 23.08.2021

1.4 Phaser 3 API Documentation - Class: Camera

https://photonstorm.github.io/phaser3-docs/Phaser.Cameras.Scene2D.Camera.html

aufgerufen am 07.09.2021

1.5 Audio - Notes of Phaser 3

https://rexrainbow.github.io/phaser3-rex-notes/docs/site/audio/

aufgerufen am 07.09.2021

1.6 A Search Algorithm - GeeksforGeeks*

https://www.geeksforgeeks.org/a-search-algorithm/

aufgerufen am 01.09.2021

8 Eidesstattliche Erklärung

Ich erkläre, dass ich die Seminararbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis aufgeführten Quellen und Hilfsmittel verwendet habe.

Ingolstadt, den dd.mm.yyyy