Skater Danger

Ein Spiel von Jarel und Cyrill

Auftrag

Auftrag war, ein Spiel mit der Jython – Bibliothek «GameGrid» zu entwickeln, mit der wir schon ein bisschen gearbeitet haben. Das Spiel sollte so kreativ wie möglich sein, korrekt Objekt orientiert programmiert sein und möglichst keine Bugs oder Softlocks haben.

Idee

Ein Mix zwischen den Games «Portal» und «Mario». Die Idee ist, mehrere Levels zu haben und das Level zu beenden soll recht einfach sein. Doch über die Level werden verschieden Coins/Gems verteilt, die extra Punkte geben und ein bisschen mehr Strategie und Puzzlesolving brauchen.
Die Grundidee für das Spiel kam vom Spiel «Do it for me», gemacht von «LixianTV»
https://www.youtube.com/watch?v=jHFipLmnjXg (Link zum Video von LixianTV)

Journal

01.12.21 Grundideen für das Spiel besprochen, als Strategie und Plattformspiel eingeordnet.

08.12.21

Ziele:

- Playerclass in einem File wo dann später nur Classen definiert sind.
- Gravitation für Player mit «Game Over» (erreicht, wenn Player unterhalb des Screens ist)
- Anfangen mit der Kollisionserkennung zwischen Player und den Groundtiles

Getan:

- Player – Class in einem File, zusammen mit Classes für Groundtiles und Game-Over
- Gravitation für Player mit «Game Over» - Mechanik
- Implementierten Spawning-Mechanismus für Groundtile
- Keine Zeit mehr für Kollisionserkennung, dafür konnten wir ein sehr modulares und flexibles System zum spawnen der Groundtiles implementieren

14.12.2021

Ziele:

- Kollision – System zwischen Player und GroundTiles endlich fertig implementieren

Getan:

- Kollision wird erkannt und der Player hat bei Kollision auch keine Gravitation mehr
  ➔ Er fällt nicht durch den Block

15.12.2021

Problem:

- Wir haben eine Liste mit einer runden Klammer statt einer eckigen Klammer abgeschlossen und 10 min das Problem gesucht.

Ziele:

- Inputs: Links, Rechts, Jump

Getan:

- Inputs gehen aber
  1. Player kann unendlich schnell werden
  2. die Gravitation ist permanent ausgeschaltet, sobald der Player eine Groundtile berührt.

20.12.2020

Ziele:

- Bugfixes, Code strukturierter machen (mit einem GameController File mit allen wichtigen Funktionen)

Getan:

- No-Gravity Bug wurde gelöst: Das Spiel nimmt an, dass der Player nach jedem Frame in der Luft ist, wenn das nicht der Fall ist, korrigiert die Colllide-Funktion die Variable isGrounded zu «True».
- Geschwindigkeits-limit eingeführt (5 Pixels pro Frame)
- GameController-file mit Funktionen «spawnTiles(tiles)» und «getInputs(Player)»

Bugs:

- Die X-Geschwindigkeit des Players bleibt im freien Fall manchmal einfach auf 3, obwohl die A-Taste für links gedrückt wird.
  Erwartet: Die X-Geschwindigkeit verkleinert sich um 1 jeden Frame bis auf -3.
- Manchmal crasht das Spiel einfach, mit der Fehlermeldung:

```
Traceback (most recent call last):
  File "C:\Users\Cyrill Marti\OneDrive - Kanton Glarus\Informatik_Game\Code\main.py", line 20, in <module>
    GC.getInputs(Player)
  File "C:\Users\Cyrill Marti\OneDrive - Kanton Glarus\Informatik_Game\Code\GameController.py", line 13, in getInputs
    if isKeyPressed(32):
  File "C:\Program Files\bin\tigerjython2.jar\Lib\gamegrid.py", line 452, in isKeyPressed
java.util.ConcurrentModificationException: java.util.ConcurrentModificationException
```

21.12.2021

- Game-crasher bug wurde gelöst: Das Programm gibt einen leeren print-output sobald eine Taste gedrückt wird, das verhindert das Auslösen einer ConcurrentModificationException, da das Spiel jetzt nach jedem input ein bisschen Zeit mit dem print «verschwendet» und deshalb nicht gleichzeitig mehrere Stellen in der Player.inputs - Array verändern kann.

Ziele:

- Groundtiles können sich in periodischen Abständen unsichtbar werden und den Player hindurchfallen lassen

Getan:

- Groundtiles können sich nun in periodischen Abständen unsichtbar machen und den Player hindurchfallen lassen

22.12.2021

Ziele:

- Hinzufügen von Münzen, die bei Kollision mit dem Player verschwinden und dem Player seine Punktzahl erhöhen.

Problem:

- Punktestand hinzufügen, sonst normales Debugging
- Game-Crasher bug (ConcurrentModificationException) trat wieder auf, allerdings nur einmal in mehreren Dutzend play-tests.

Getan:

- Münzen hinzugefügt
- Hinzufügen von Punktestand

Wegen Verwechslungsgefahr zwischen der Instanz Player von der Klasse Player im File «class_definitions.py» wurde die Klasse Player in ein eigenes File verschoben (Player.py)

Ziele für Weihnachtsferien:

- Pressure plates
- Designen von Sprites

29.12.2021

- Im Spiel gibt es jetzt Pressure plates. Wenn der Player das erste Mal auf eine draufsteht, erscheinen neue Tiles und einige werden gelöscht. Das Level kann also vom Player selbst geändert werden. Damit das funktioniert verlangt die PressurePlate-class zwei Argumente nebst der Position der Pressure plate: Eine Liste von Tiles die sie spawnen sollte und eine Liste von Tiles die sie zerstörten sollte. Um die Tiles zu spawnen ruft die Pressure plate die früher schon definierte «spawnTiles» - Methode auf, um die Tiles zu löschen ruft es die neue Funktion «despawnTiles» auf.

06.01.2022

- Neue Sprites (by Jarel) sind nun im Game. Die Münze besitzt 10 verschiedene Sprites für eine Spritesheet – Animation. Die «instabilen» Groundtiles haben jetzt Risse

07.01.2022

- ConcurrentModificationException ist (hoffentlich) gelöst. Die Funktion avoidCME() wird nach jedem Input aufgerufen und gibt das Produkt von zwei zufällig gewählten Integern zwischen 500 und 1500 zurück --> Die Zeit, die ein Thread braucht um die Multiplikation auszuführen

ist (fast) immer unterschiedlich --> Kein Thread kann gleichzeitig mit einem anderen auf eine Liste zugreifen.

11.01.2022

Ziele:

- LevelLoader Funktion, um ein beliebiges Level zu laden.
- Levels in ein «Dateiformat» umändern, das von «LevelLoader» aktzeptiert wird.

Getan:

- LevelLoader hinzugefüg,noch ein paar Bugs drin

12.01.22

- Bugs beseitigt

Ziele:

- Mit den Levels anfangen

Getan:

- 1.Level

21.01.2022

- ConcurrentModificationException ist ein für alle Mal gelöst. Das Programm importiert die Definiton für eine CME (import java.util.ConcurrentModificationExcption as CME) und verwendet danach die python built-in Clause «try-except» um die Exception einzufangen und verhindert so ein Programmcrash

23.01.2022

- Die letzten paar Levels wurden einprogrammiert. Auch laufen nach dem letzten Level die Credits und es wird ausgegeben, wie viele Coins der Player über die Levels gesammelt hat (max. 8)

Fazit

Das Projekt war definitiv eine gelungene Abwechslung zum Informatik-Unterricht. Das Spiel kam nach unserer Meinung nach nicht einmal so schlecht raus. Was uns überrascht hat, war, wie nervig GameGrid sein kann. Obwohl es oberflächlich nach einer einfachen, doch immer noch leistungsfähiger Bibliothek aussieht, hat es mehrere Bugs und Fehler, die man mit dem Python – Code einfach nicht lösen kann (ConcurrentModificationException tritt nach sage und schreibe 5 Patches endlich nicht mehr auf). Trotzdem haben wir für die meisten dieser Probleme Lösungen gefunden. Natürlich hat unser Game einige Bugs und Probleme, da 2 Monate einfach nicht genug sind für ein problemfreies Game. Der Bug, der am schnellsten auffallen sollte, ist, dass der Player einfach irgendwo ein Block berühren muss, dass er nicht herunterfällt. Diesen Bug wird wohl einfach für immer bleiben (vielleicht benennen wir ihn zu einem Feature um).

Auch ist unser Game nicht das originellste oder raffinierteste. Doch dank der Pressure Plate – Mechanik konnten wir doch noch eine (mehr oder weniger) originelle Idee im Game verwirklichen.

Wir glauben, dass wir uns als Team perfekt ergänzt haben. Jarel war sehr kreativ unterwegs, hat viele Ideen geliefert und auch alle Sprites, die im Game verwendet werden gezeichnet. Cyrill wiederum hat diese Ideen – mit Hilfe von Jarel – in Code umgesetzt. Ohne die Zusammenarbeit wäre das Game wohl nicht annähernd so weit entwickelt wie es im Moment ist.

Motivation

Unsere Motivation entwickelte sich vom Anfang an konstant nach oben. Sie wurde definitiv gedämpft durch den Kampf gegen die ConcurrentModificationException und all die anderen Bugs, die sich in unserem Game eingenistet haben. Gegen Ende kam zum Glück der beste Part: Level Design (Whoo)! Doch das Einprogrammieren der Levels in das „levels.py“ – File war doch recht monoton und anstrengend. Aber generell waren wird die ganze Zeit sehr motiviert.

Funktionsweise

Unser Game-Code ist in mehrere Files aufgeteilt, die untereinander kommunizieren können. Diesen Weg haben wir eingeschlagen, um den Code übersichtlicher zu machen. So sind alle Klassendefinitionen (mit Ausnahme des Players) im File «class_definitions.py», alle wichtigen, spielsteuernden Funktionen im File «GameController.py» und alle Levels im File «levels.py». Das «main.py» File ist nur 10 Zeilen lang und macht nur das Setup des Bildschirms und startet das Game. Nachher übernimmt das File «GameController.py» die Arbeit.

Dieser Abschnitt sollte Ihnen einen Einblick geben, wie wir das Game aufgebaut haben, was wo geregelt wird und wieso wir das gemacht haben, da der Code (trotz Kommentaren) auf Anhieb vielleicht schwierig zu verstehen ist.

Player:

Das unbestreitbar wichtigste Element in unserem Game ist der Player. Die Klassendefinition für Player finden Sie im File «Player.py». In jedem Level gibt es immer genau eine Instanz der Klasse Player. Die zwei wichtigsten Variablen des Players sind die zwei Listen «self.speed» und «self.inputs». Die «self.inputs» - Liste wird in jedem Frame von der Funktion getInputs() (GameController.py, Zeile 27) verändert. Das erste Element der Liste wird von False zu True, wenn die A-Taste gedrückt wird und springt zurück zu False, wenn sie losgelassen wird. Dasselbe mit dem zweiten und dritten Element in der Liste. Das zweite bei der D-Taste und das dritte beim Drücken der Space-Taste.

GroundTiles

Die GroundTile - Klassendefinition finden Sie im File „class-definitions.py“ ab Zeile 5. In jedem Level gibt es mehrere Instanzen von GroundTile. Eine GroundTile ist definiert durch drei Parameter: Location, isFlashing, flashingPeriod. Der erste Parameter gibt an, wo im Spielfeld die GroundTile erscheinen soll, der zweite gibt an, ob die GroundTile periodisch verschwindet und den Player dann zu seinem Tod fallen lässt. Der dritte Parameter gibt an, wie lange die Dauer zwischen Verschwinden und wieder Auftauchen ist («None» falls isFlashing «False»).

Pressure Plate

Die PressurePlate – Klassendefinition finden Sie im File «class-definitions.py» ab Zeile 38. In jedem Level gibt es zwischen 1-5 PressurePlates. Eine PressurePlate braucht folgende Parameter: Location, tilesToSpawn, tilesToDespawn. Der erste Parameter siehe oben. Der zweite Parameter ist eine Liste aus GroundTiles (siehe oben), die, wenn die PressurePlate «gedrückt» wird, erscheinen werden. Die GroundTiles in der Liste tilesToDespawn werden gelöscht, sobald die PressurePlate aktiviert wird. In jedem Level gibt es auch eine goldene PressurePlate, die, wenn sie gedrückt wird, den Player ins nächste Level teleportiert.

Coins

In jedem Level gibt es mehrere Coins. Deren Klassendefinition finden Sie im File «class-definitions.py» ab Zeile 26. Ein Coin ist definiert durch einen Parameter: Location. Der Coin macht eine Sprite-Sheet-Animation mit 10 Bildern und kann vom Player «aufgelesen» werden.

CollisionListeners

Im File «class-definitions.py» ganz unten finden Sie all CollisionListeners, die im Spiel verwendet werden. Der Player hat den CollisionListener **«PlayerCL»**, der nach einer Kollison zwischen dem Player und einer GroundTile sucht. Wenn eine Kollsion stattfindet, wird der Player nicht mehr nach unten beschleunigt. Der CollisionListener **«PressurePlatePlayerCL»** wird den PressurePlates zugewiesen und sucht nach einer Kollision zwischen der PressurePlate und dem Player. Wenn eine solche stattfindet, wird die PressurePlate «gedrückt» und führt die oben (in PressurePlate) beschriebenen Aktionen aus. Der CollisonListener **«CoinPlayerCL»** wird jedem Coin zugewiesen und sucht nach einer Kollision zwischen dem Coin, dem er zugewiesen wurde und dem Player. Wenn er eine Kollision erkennt, löscht sich der Coin und der Punktestand des Players wird um eins erhöht.

Persönlicher Kommentar

Natürlich, der Code ist in keinster Weise perfekt OOP (Object Oriented Programming). Aber in meiner (Cyrills) Meinung ist Python nicht die Sprache, in der man alles mit OOP lösen sollte. Für das wäre Java die perfekte Lösung gewesen. Den Code neuzuschreiben und alle Funktionen des «GameController.py» - Files in eine «GameController» - Klasse mit «staticmethods» zu schreiben und dann alles vom «main.py» - File aus zu steuern erschien mir überflüssig, da der Code bereits funktionierte und auch mit den Master-Funktionen aus «GameController» recht übersichtlich war. Die Sprache in dieser Dokumentation lässt wohl auch zu wünschen übrig. Natürlich hätten wir «Druckplatte» und «Bodenziegel» für «Pressure plate» und «Ground tile» verwenden können, doch da der Code komplett auf Englisch ist und die meisten Begriffe in der Spieleentwicklung auf Englisch sind, haben wir uns entschlossen, englische Begriffe recht häufig zu verwenden.