ZHAW School of Engineering, IT17ta_ZH, PSIT 4

Projekt Darwin

Michael Schaufelberger (Product Owner)
Filip Kašiković (Scrum Master)
Raphael Mailänder
Simon Stucki
Marc Berli
Ferenc Kuntič

6. Mai 2020

Abstract

In der heutigen Zeit haben sich Videospiele in unserer Gesellschaft stark etabliert und sind ein Teil unseres Alltags geworden. Die Branche und Präsenz der Spieleindustrie wächst von Jahr zu Jahr stetig an und bildet ein sehr lukratives Geschäftsumfeld. Erträge und Umsätze in Milliardenhöhe sind keine Seltenheit. Unternehmen mit mehreren hundert Mitarbeitern, versuchen im Schnelltempo Spiele verschiedenster Art auf den Markt zu bringen. Angesichts solch grosser Konkurrenz stellen sich uns die Fragen: Wie gestalten wir ein attraktives Spiel? Wer ist die Zielgruppe? Wie können wir die Spieler für das Spiel begeistern? Welche Qualitäten braucht das Spiel? Um all diese Fragen beantworten zu können, braucht es ein fundiertes Konzept, durchdachte Überlegungen und ein gutes Team. Obwohl die Konkurrenz sehr gross ist, sehen wir eine Möglichkeit, mit einem interaktiven Mehrspieler-Spiel den Durchbruch zu schaffen.

Ziel dieser Arbeit ist es, ein innovatives und erfinderisches Spiel zu entwickeln, das sowohl logisches Denken wie auch technisches Interesse fördert. Das Ziel des Spieles ist es, Spieleinheiten mit kleinen Code-Abschnitten zu steuern und schlussendlich der letzte ”Überlebende” zu sein.

Bei der Entwicklung werden modernste Technologien eingesetzt. Für den Frontend-Bereich haben wir uns für React entscheiden. Dies ermöglicht es uns die Applikation in Komponenten zu entwickeln und erhöht den Wiederverwendungsgrad. Für die Realisierung des Backends setzen wir auf Node.js und TypeScript, da es uns erlaubt die Applikation modularer zu bauen und sie eine grosse Community besitzen. Für das Projektmanagement der Entwicklung wurde uns Scrum als agilen Prozess vorgegeben.

Das Ergebnis des ”Projekt Darwin” ist ein herausforderndes Überlebensspiel, in dem mehrere Spieler gegeneinander antreten können.

1 Einleitung

Bestehende Spiele, die mittels Programmierung gesteuert werden, zielen sehr stark darauf ab die Programmierfähigkeiten zu verbessern, oder ganz grundlegend, die beim Programmieren notwendige Art des Denkens zu vermitteln. Die meisten dieser Spiele sind für Kinder gemacht, da diese möglichst spielerisch dazu gebracht werden wollen, Dinge zu erlernen. Allerdings wissen auch grössere Kinder eine spielerische Lernweise zu schätzen.
Bei vielen der Lösungen steht das Programmieren im Vordergrund - die Grafik und damit auch der Spielspass, lassen zu wünschen übrig. Weiter sind die meisten der Konkurrenzprodukte nur für einen Spieler ausgelegt (”Single Player”). Auch dies limitiert den Spielspass. Um diese Schwächen zu adressieren, haben wir ”Projekt Darwin” entwickelt.

1.1 Zielsetzungen

Ziel von ”Projekt Darwin” ist es, sowohl den Spielspass zu maximieren, als auch die algorithmische, logische Denkweise zu fördern. Das Spiel soll sowohl von Kindern als auch von Erwachsenen gespielt werden können, wobei die Grafik insbesondere nicht kindlich sein soll. Das Spiel in seiner jetzigen Version lässt sich mit wenigen Befehlen spielen, weshalb nur minimale, technische Vorkenntnisse nötig sind. In einer zukünftigen Version könnte man das verwendete Set an Befehlen in höheren Levels erweitern, sodass die Komplexität zunimmt und der Spieler im Laufe der Zeit dazulernen kann.
Ein weiteres Ziel des Projektes ist es, die Kompetenzen des Entwicklungsteams bezüglich agiler Software-Entwicklung aufzubauen und zu festigen.

2 Resultate

In diesem Kapitel werden die Resultate und Ergebnisse dieses Projektes präsentiert. Es wird auf die einzelnen Bausteine der Applikation detaillierter eingegangen und die Lösungsentscheide beschrieben, die innerhalb des Teams getroffen wurden.

2.1 Technische Design-Entscheidungen

Für die Umsetzung des Projekts waren verschiedene technische Entscheidungen nötig, die im Folgenden erläutert werden.

2.1.1 TypeScript / Node.js

TypeScript ist eine Programmiersprache, die von Microsoft entwickelt wurde und Code in JavaScript kompiliert. Durch die Typisierung ermöglicht sie es, verschiedene Module besser zu organisieren und Refactoring einfacher durchzuführen. Da es sich bei Darwin um eine Webapplikation handelt, welche in Frontend und Backend aufgeteilt ist, und aufgrund der Erfahrung der Teammitglieder, ist die Entscheidung gefallen, ausschliesslich TypeScript einzusetzen. Mithilfe von TypeScript kann der Quellcode mit einer besseren Struktur versehen werden (siehe zum Beispiel die Struktur der Spielobjekte in Abb. 1) und ihn besser formulieren sowie auch einfacher skalieren.

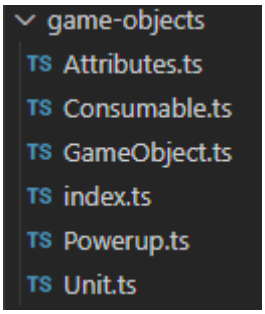

Abbildung 1: Datei- und Ordnerstruktur der Spielobjekte

Mittels Node.js kann sowohl im Frontend als auch im Backend JavaScript eingesetzt werden. Dies ermöglicht es uns, Code einfacher zwischen den verschiedenen Teilen der Applikation wiederzuverwenden. Zudem ist es eine sehr beliebte JavaScript Runtime [1] und weist die notwendige Stabilität für das Projekt Darwin auf.

2.1.2 React / 2D Canvas

React ist eine JavaScript-Bibliothek, deren Ziel es ist, die Entwicklung von grafischen Benutzeroberflächen im Webbrowser zu vereinfachen. Sie wurde 2013 von Facebook entwickelt [2] und ist in der Web-Entwicklung mittlerweile stark verbreitet. Aufgrund der bestehenden Erfahrung des Teams und einer sehr grossen Community, fiel der Entscheid leicht, React zu benutzen. Für die Visualisierung des Spiels und dessen Objekte wird ein 2D Canvas mittels Pixi.js eingesetzt. Der Einsatz von Pixi.js erlaubt es, die Canvas API zu abstrahieren und somit den Aufwand für die Umsetzung der Visualisierung zu verringern.

2.1.3 Lerna / Monorepo

Lerna ist ein Tool, das die Verwaltung von Repositories mit mehreren Paketen mittels Git und NPM, dem Paketmanager von Node.js, optimiert [3]. Bei diesem Projekt hilft Lerna mit der Verwaltung von Paketen, die sowohl im Frontend als auch im Backend verwendet werden.

2.1.4 Testing / Jest

Unit Tests werden automatisch bei jedem Build ausgeführt. Als Framework wird Jest von Facebook verwendet. Dieses ist in der JavaScript-Community weit verbreitet und im Team ist bereits Know-How vorhanden. Beim Unit Testing werden einzelne Einheiten isoliert getestet und sowohl im Frontend als auch im Backend separat ausgeführt.

2.1.5 Workflow / CI|CD

Ein reibungsloser Entwicklungsprozess hat für das Entwicklungsteam oberste Priorität. Deswegen wurde beschlossen eine Build- sowie eine Releasepipeline einzurichten (siehe Abb. 2). Diese ermöglicht es, Pull-Requests automatisch zu testen und bei einem erfolgten Merge automatisiert zu Deployen.

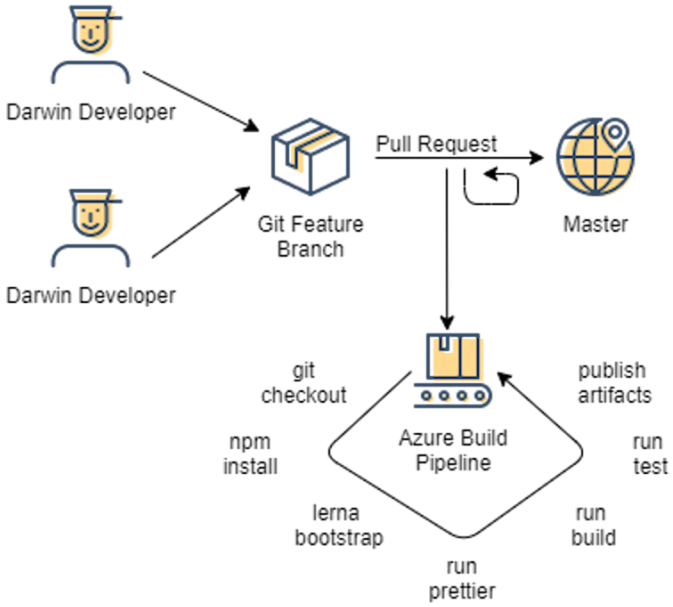

Abbildung 2: Entwicklungsprozess und CI/CD-Pipeline

2.2 Implementiertes Spiel

In diesem Abschnitt werden die Elemente des implementierten Spiels beschrieben.

2.2.1 Benutzeroberfläche

Wie in Abbildung 3 ersichtlich ist, gliedert sich die Benutzeroberfläche in ein Spielfeld (rechts) und ein Eingabefeld (links), durch das der Spieler den Spielverlauf steuern kann. Innerhalb des Spiels ist eine detaillierte Hilfeseite mit Beispielen direkt unter dem Spielfeld einsehbar.

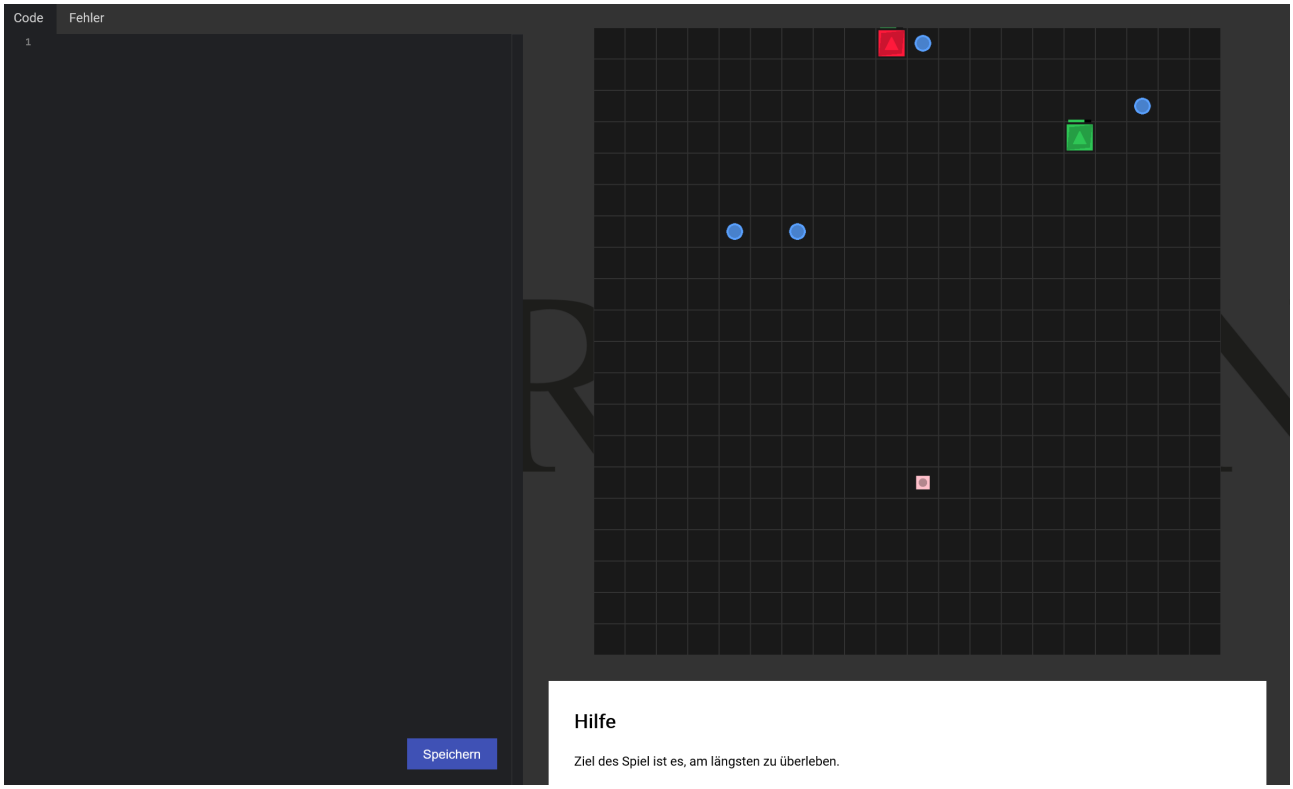

Abbildung 3: Layout der Benutzeroberfläche

2.2.2 Spielfeld

Das Spielfeld (sichtbar in Abb. 4) ist in ein Raster unterteilt. Die Spielobjekte werden als geometrische Figuren visualisiert. Diese geometrischen Formen haben folgende Bedeutung:

- Grünes Quadrat: Der aktuelle Spieler (”ich”)
- Rotes Quadrat: Anderer Spieler (”Gegner”)
- Balken über den Quadraten: Gesundheitszustand des Spielers (”Health”)
- Blauer Kreis: Nahrung
- Kleine Quadrate (verschiedene Farben): ”Power-Ups”, die Spieler temporär beeinflussen, falls diese konsumiert werden.

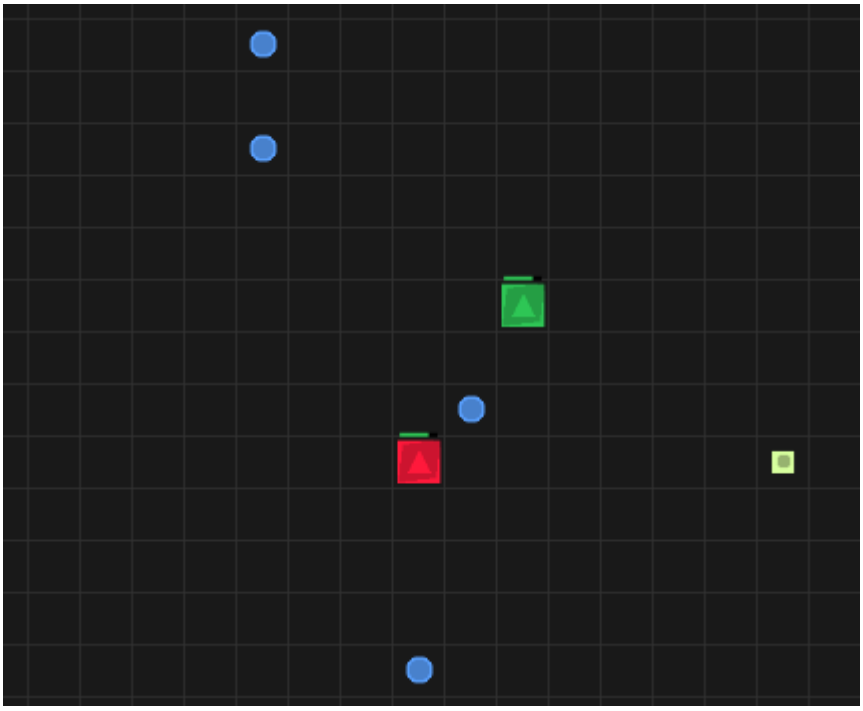

Abbildung 4: Design Spielfeld

2.2.3 Spielführung

Gespielt wird mittels Code-Eingabe, zum Beispiel `move('LEFT')`, wie in Abbildung 5 ersichtlich. Ziel ist es, als Letzter noch zu leben. Um am Leben zu bleiben, muss man Nahrung sammeln. Das Konsumieren von Powerups kann dem Spieler einen entscheidenden Vorteil verschaffen, um den Sieg zu erlangen. Ist man gerüstet, kann der Spieler auch gegnerische Spieler angreifen, um dem Sieg ein Stück näher zu kommen. Aber hierbei ist Vorsicht geboten: Dieses Manöver kann nach hinten los gehen!

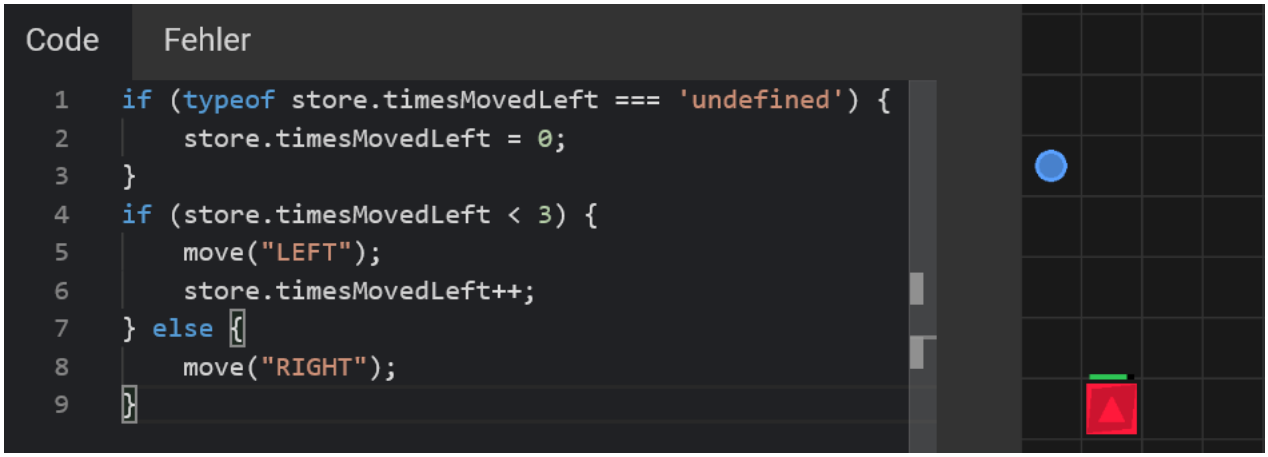

Abbildung 5: Design Code-Eingabe

3 Diskussion und Ausblick

Projekt Darwin wurde gemäss unserer Planung erfolgreich umgesetzt. Wir haben alle Funktionen implementiert und können ein lauffähiges Spiel präsentieren.

3.1 Workflow

Die automatisierte Build-Pipeline hat sich als sehr wichtig herausgestellt, um während der Entwicklungsphase mit mehreren Entwicklern effizient arbeiten zu können. Der Workflow gewährleistet, dass ausschliesslich funktionierender Quellcode veröffentlicht wird. Dies hat den Vorteil, dass nachfolgende Entwickler nicht blockiert werden, falls eingecheckter Code nicht kompiliert. Auch das Testing kann deshalb verzögerungsfrei starten, da es nicht auf eine nicht funktionierende Umgebung warten muss.

3.2 Benutzeroberfläche

Die einfach gehaltene Grafik und das schnörkellose Design unterstützen den Spieler dabei, sich auf das Wesentliche zu konzentrieren. Die angebotene Hilfe mit Beispielen erweist sich als sehr nützlich, insbesondere für Programmier-Anfänger. Schliesslich soll man durch das Spiel auch lernen, zu programmieren.

3.3 Spielfeld

Die Grafik der Icons auf dem Spielfeld war zu Beginn etwas aufwendiger geplant, das Team wollte Figuren aus der ”Wilhelm Tell”-Sage benutzen. Wir haben uns jedoch früh dagegen entschieden, komplexe Icons zu entwerfen. Die aufwendige Darstellung hätte dem Spiel zwar ein interessantes Thema gegeben, jedoch durch den höheren Aufwand andere Funktionen des Spiels verzögert. Die jetzt verwendeten geometrischen Formen sind sehr klar, für den Spieler gut erkennbar und passen auch insgesamt gut zur Grafik. Wir sind deshalb mit dieser Anpassung zufrieden.

3.4 Spielführung

Die Regeln des Spiels sind sehr einfach gehalten, das hilft dem Spieler, sich auf das Hauptziel zu konzentrieren, nämlich das Programmieren zu erlernen und seine Strategie in Code umzusetzen.

3.5 Scrum

Scrum ermöglicht, den Entwicklungsprozess strukturiert und effizient zu gestalten. Ein agiles Umfeld wie Scrum hilft, die auftretenden Probleme besser und schneller zu bewältigen. Wir sind deshalb froh, dass wir dieses Framework benutzt haben.

3.6 Ausblick

Es gibt diverse Erweiterungsmöglichkeiten für das Spiel. Einige werden im Folgenden näher beschrieben:

3.6.1 Benutzeroberfläche

Es ist möglich, das ursprünglich angedachte, komplexere Icon-Set zu implementieren. Der Spieler könnte sich damit vielleicht besser mit der Geschichte identifizieren. Dies hat für den Product Owner allerdings im Moment keine Priorität, da der Aufwand für die Erstellung des Icon-Sets, im Vergleich zum Nutzen, zu gross ist.

3.6.2 Befehlssatz

In Zukunft können die zur Verfügung stehenden Befehle erweitert werden, zum Beispiel durch weiterführende Levels, sodass auch neue Strategien möglich werden. Damit könnte man zum Beispiel die Spieler durch verschiedene Lernstufen begleiten und sie können ihre Fähigkeiten Schritt für Schritt verbessern.

3.6.3 Spielmodus

Es ist auch möglich, verschiedene neue Spielobjekte mit einzuführen. Zum Beispiel solche, die man sammeln kann oder denen man ausweichen muss. Es besteht aber die Gefahr, dass das Spiel dadurch unnötig verkompliziert wird, ohne dass der Spieler seine Programmierfähigkeiten verbessert. Daher müssen diese Anpassungen sehr vorsichtig vorgenommen werden.

Literatur

[1] F. Copes. (2020-04-03). “Introduction to Node.js”, Adresse: `https://nodejs.dev/` (besucht am 2020-04-30).

[2] F. Copes. (2019-01-08). “The React Handbook”, Adresse: `https://www.freecodecamp.org/news/the-react-handbook-b71c27b0a795/` (besucht am 2020-04-18).

[3] J. George. (2019-12-15). “Lerna, A tool for managing JavaScript projects with multiple packages”, Adresse: `https://github.com/lerna/lerna/blob/master/README.md` (besucht am 2020-04-18).

Abbildungsverzeichnis