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.
Im Angesicht solch grosser Konkurrenz stellt sich uns die Frage, 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 Multiplayer-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 funktionierendes Spiel, in dem mehrere Spieler gegeneinander antreten können.

1 Einleitung

1.1 Ausgangslage

Bestehende Games, die mittels Coding gesteuert werden, zielen sehr stark darauf ab die Coding-Skills zu verbessern, oder ganz grundlegend die beim Programmieren notwendige Art des Denkens zu vermitteln.
Die meisten dieser Games 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 der Code im Vordergrund und 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.

1.2 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 keine ”kindliche” (z. B. sehr bunt) sein soll.
Das Spiel in seiner jetzigen Version lässt sich mit wenigen Befehlen spielen, weshalb die technischen Vorkenntnisse minimal sein können.
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 dazu lernen kann.
Ein weiteres Ziel des Projektes war es, die Kompetenzen des Entwicklungsteams bzgl. agiler Software-Entwicklung aufzubauen und zu festigen.

2 Resultate

In diesem Kapitel werden die Resultate und Ergebnisse unseres Projektes präsentiert. Es wird auf die einzelnen Bausteine unserer Applikation detaillierter eingegangen und die Lösungsentscheide beschrieben, die wir innerhalb des Teams getroffen haben.

2.1 Technische Design-Entscheidungen

2.1.1 TypeScript / Node.js

TypeScript ist eine Programmiersprache, die von Microsoft entwickelt wurde. Durch die Typisierung ermöglicht es uns, verschiedene Module einfacher zu überarbeiten.
Da es sich bei Darwin um eine Webapplikation handelt, welche in Frontend und Backend aufgeteilt ist, haben wir uns aufgrund der Erfahrung der Teammitglieder für das ausschliessliche Einsetzen von TypeScript bzw. JavaScript entschieden. Mithilfe von TypeScript können wir unseren Quellcode mit einer besseren Struktur versehen (siehe z. B. die Struktur der Spiel-Objekte in Abb. 1) und ihn besser formulieren, sowie auch einfacher skalieren.

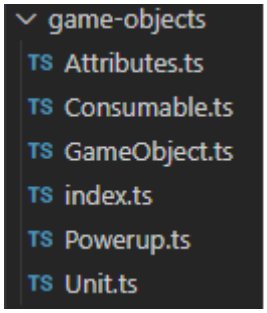

Abbildung 1: Erstellen von Spiel-Objekten mittels TypeScript / Darwin Types

Mit Node.js können wir nicht nur auch im Backend JavaScript einsetzen. 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 User Interfaces 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 uns der Entscheid React zu benutzen leicht.
Für die Visualisierung des Spiels und dessen Objekte setzen wir 2D Canvas mittels Pixi.js ein. Der Einsatz von Pixi.js bietet uns die Möglichkeit, die Canvas API zu abstrahieren und sich somit die Umsetzung der Visualisierung weniger aufwändig gestaltet.

2.1.3 Lerna / Monorepo

Lerna ist ein Tool, das die Verwaltung von Repositories mit mehreren Paketen mittels git und npm optimiert [3]. Bei unserem Projekt hilft uns Lerna mit der Verwaltung von Paketen, die im Frontend

und Backend verwendet werden.

2.1.4 Testing / Jest

Unit Tests werden automatisiert bei jedem Build und Pull-Request ausgeführt. Als Framework wird Jest von Facebook verwendet. Dieses ist in der Javascript-Welt sehr 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 uns oberste Priorität, deswegen haben wir 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.

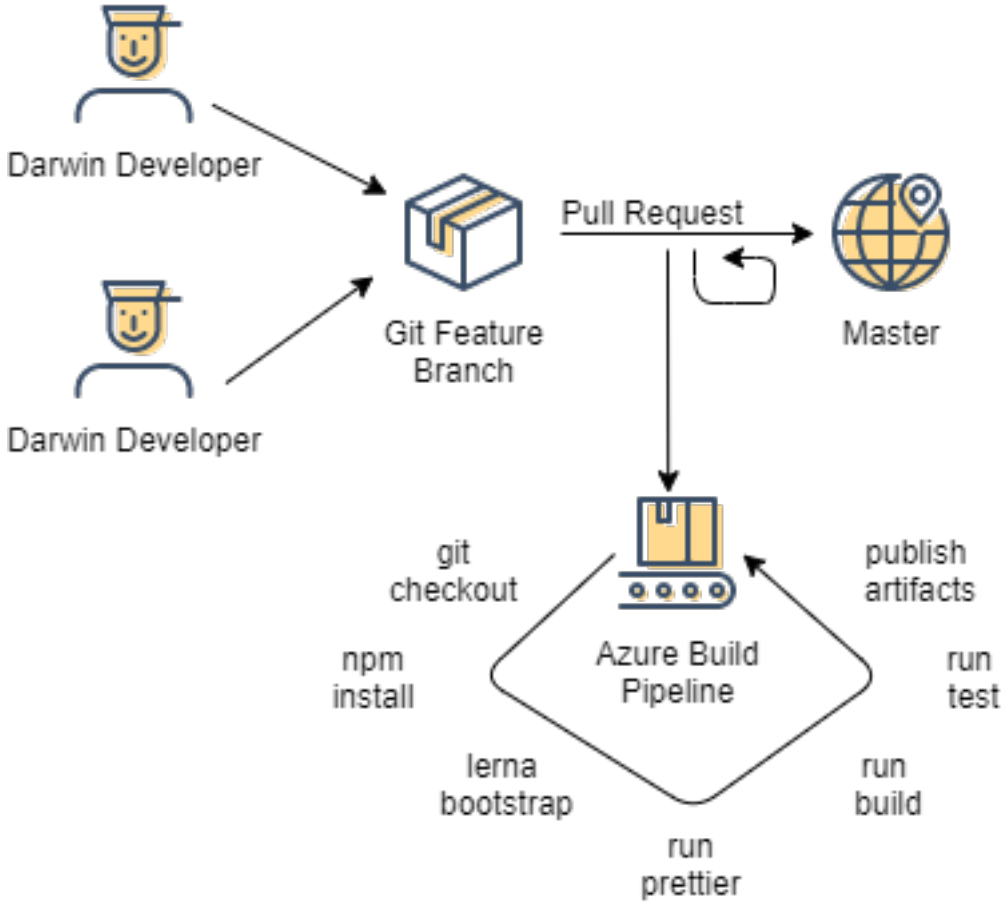

Abbildung 2: Design Entwicklungsprozess / CI CD

2.2 Implementiertes Spiel

In diesem Abschnitt werden die Elemente des implementierten Spiels beschrieben.

2.2.1 UI

Wie in Abbildung 3 ersichtlich, gliedert sich das UI in ein Spielfeld (rechts) und eine Textinput-Area (links), durch die der Spieler den Spielverlauf steuern kann. Es ist innerhalb des Spiels eine detail-

lierte Hilfeseite mit Beispielen direkt unter dem Spielfeld einsehbar.

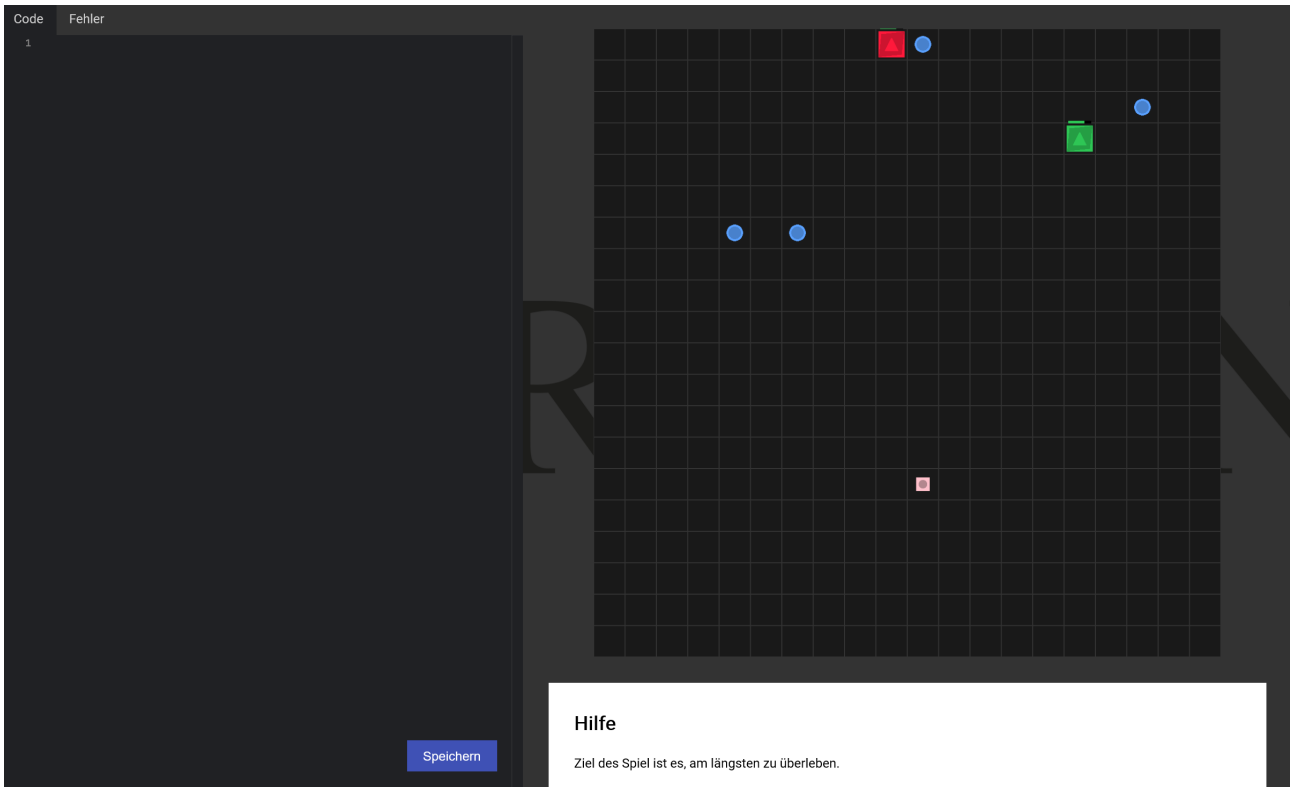

Abbildung 3: Übersicht Design Darwin Spiel

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”

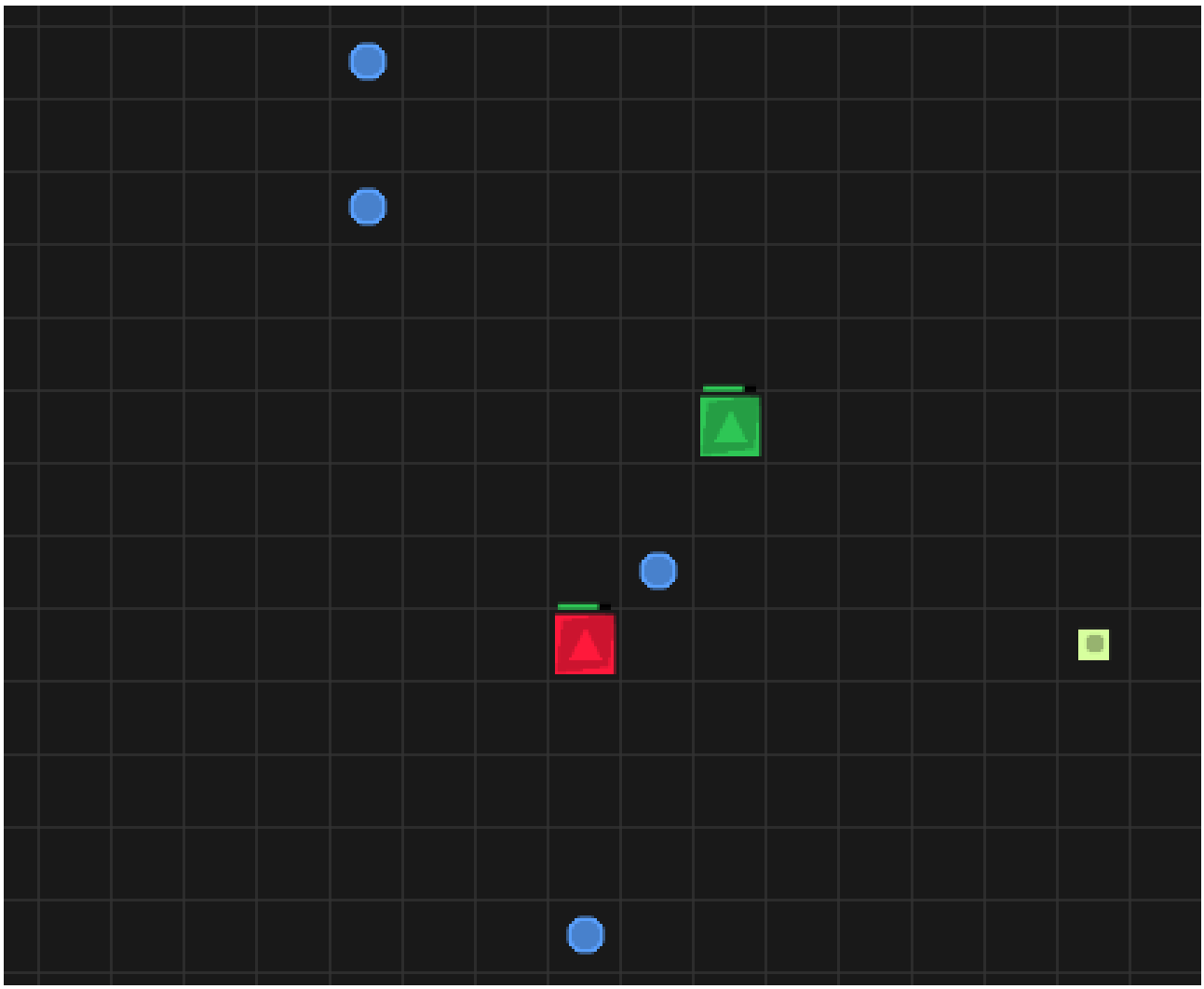

Abbildung 4: Design Spielfeld

2.2.3 Spielführung

Gespielt wird mittels Code-Eingabe, z.B. `move('LEFT')`, wie in Abbildung 5 ersichtlich. Ziel ist es, als Letzter zu überleben. Um am Leben zu bleiben, muss man Nahrung sammeln. Das konsumieren von Powerups kann einem einen entscheidenden Vorteil im Kampf um den Siegverschaffen. Ist man gerüstet, kann der Spieler auch gegnerische Spieler angreifen, um dem Sieg ein Stück näher zu kommen. Aber hierbei ist Vorsicht geboten! Dieses Maneuver kann nach hinten los gehen.

Abbildung 5: Design Code-Eingabe

3 Diskussion und Ausblick

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 Code deployed wird. Das führt dazu, dass weder nachfolgende Entwickler warten müssen, um Code einzuchecken (weil z.B. gerade jemand den Build ”gebrochen” hat mit seinem Check-in), noch das Testing wegen einer nicht funktionierender Umgebungen warten muss.

3.2 UI

Die einfach gehaltene Grafik und das schnörkellose Design unterstützt den Spieler dabei, sich auf das Wesentliche zu konzentrieren. Die angebotene Hilfeseite mit Code-Beispielen erweist sich als sehr hilfreich, 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 designen. Die aufwendige Darstellung hätte dem Spiel zwar ein interessantes Thema gegeben, jedoch inhaltlich keinen Mehrwert geliefert. 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 konzentieren, nämlich das Coden 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 half uns, 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 sind im Folgenden näher beschrieben:

3.6.1 UI

Wir könnten uns überlegen, das ursprünglich angedachte, komplexere Iconset zu implementieren. Der Spieler könnte sich damit vielleicht besser mit der Story identifizieren. Dies hat für den Product Owner allerdings im Moment keine Priorität, da der Aufwand für die Erstellung des Iconsets sehr gross wäre im Vergleich zum Nutzen.

3.6.2 Befehlsset

In Zukunft könnten die zur Verfügung stehenden Kommandos erweitert werden, z.B. durch weiterführende Levels, sodass auch neue Strategien möglich werden. Damit könnte man z.B. die Spieler durch verschiedene Lernstufen begleiten und sie würden ihr Können Schritt für Schritt verbessern.

3.6.3 Spielmodus

Wir könnten uns vorstellen, verschiedene neue Gem-Types mit neuen Bedeutungen einzuführen. Z.B. 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 notwendigerweise verbessert. Daher müssen wir diese Anpassungen sehr vorsichtig vornehmen.

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