Inhaltsverzeichnis

Projektdokumentation „Malefiz“

Layout der Seiten/Ansichten

Das Layout der Seite bzw. die Ansicht habe ich über Styles festgelegt. Diese werden zum Teil aus dem Stylesheet heraus geladen, aber auch über die Javascript-Datei, durch verschiedene Zustände heraus, erzeugt.

Um Elemente anzuzeigen bzw. zu verbergen, wurde zum größten Teil die CSS-Eigenschaft „display: block“ als auch „display: none“ verwendet.

Einsatz der SVG- bzw. Canvas-Grafikelemente

In diesem Programm wurde SVG verwendet. Der Einsatz von SVG kommt in vielen verschiedenen Bereichen zum Vorschein.

In der Lobby werden die Buttons als auch die DIVs über SVG erzeugt. Desweiteren kommt SVG beim erzeugen des Spielfeldes zum Einsatz. Der Würfel ist ein DIV, welches über SVG, je nach gewürfelter Zahl, sein Source Attribut verändert.

Canvas kam nicht zum Einsatz.

Nachrichtenverschlüsselung

Die Nachrichtenverschlüsselung erfolgt über die Standardkodierung von base64. Jedes einzelne Zeichen wird in seinen ASCII-Zahlenwert umgewandelt und dieser Wert wird um 12 vergrößert. Dieser Wert wird wieder in ein ASCII-Zeichen gewandelt. Auf diese Weise kann nicht auf direktem Wege die Verschlüsselung gehackt werden.

So wird das Paket durch das Netzwerk geschickt. Wenn Niemand weiß, dass es so kodiert ist, kann nichts aus dieser Zeichenkette erfasst werden.

Um das ganze wieder zu decodieren, wird es wieder in den ASCII-Zahlenwert umgewandelt und dieser Wert wird dieses Mal um 12 verringert. Das Zeichen wird wieder zurück in ein ASCII-Zeichen gewandelt.

Nachrichten zur Kommunikation mit den SpielpartnerInnen und den anderen Clients des eigenen Typs

Um Nachricht durch Netzwerk zu senden und diese auch bei den anderen User anzuzeigen habe ich ein WebSocketServer verwendet. Über den ist es möglich Datenpakete in das Netzwerk zu senden. Nachdem diese Datenpakete ins Netzwerk geschickt werden, empfängt das Programm in erster Linie alle Datenpakete die über den eingetragenen WebSocketServer gesendet werden. Aber es werden nur diese ausgewertet, die ein Präfix am Nachrichtanfang haben, der beim verschicken eines Datenpaketes mit gesendet wird. Hat das Paket am Nachrichtenanfang alles andere als diesen Präfix, wird diese Paket nicht weiter verwendet, es sei denn es ist eine Errormeldung oder es konnte keine Verbindung zu diesem Server hergestellt werden. In diesem Programm sorgt eine Spielnummer dafür,

dass sich Spieler untereinander Unterhalten können oder Benutzer die kein gestartetes Spiel haben, sich untereinander Unterhalten können.

Verfahren zur Wahl und Festlegung der SpielpartnerInnen

Nachdem ein Benutzer ein Spiel angelegt hat. Wird ein Objekt erzeugt, dieses Werten alle Benutzer aus, die zu diesem Zeitpunkt auf der Lobbyseite waren. Bei jedem User ist wurde ein Button erzeugt. Mit anklicken des Button erfolgt eine Betretungen in das jeweilige Spiele. Sobald ein User in dieses Spiel beigetreten ist, wird ein neues Objekt erzeut. Dieses Objekt beinhaltet Informationen des Spielanlegers. Außerdem wird die Spieleranzahl, bei jedem Userbeitritt, um einen erhört. Sobald minimal 2 Spieler, also Anleger und ein anderer User dieses Spiel beigetreten sind (Anleger ist automatisch beim Anlegen des Spieles, in diesem Spiel), wird ein Button beim Anlege erzeugt. Über diesen Button kann das Spiel gestartet werden.

Verwendeten Datenstrukturen, Datenaustauschformate (in JSON oder XML) und Algorithmen

Die Datenstruktur Spielfeld wird mittels eines Array erzeugt. Sowohl die Anordnung der Spielfiguren und deren Koordinaten stehen in einem Array als auch eine ähnliche Kopie des Array. In dieser Kopie stehen Objekte.

Das Datenaustauschformat JSON habe ich in diesem Programm eher Betracht gezogen, da sich mit JSON-Objekte schnell erzeugen lassen und keine weiteren Dateien dafür ausgewertet werden müssen.
Um ein JSON-Objekt zu erzeugen legt man ein Javascript Objekt an und dieses und wird durch die Funktion „JSON.stringify“ zu einem JSON-Objekt umgewandelt und diese Objekt wird ins Netzwerk gesendet.

Das Programm beinhaltet viele verschiedene Algorithmen, ein habe ich bereits unter dem Punkt „Nachrichtenverschlüsselung“ vorgestellt und ein zweiter umfangreich Algorithmus wird zum auswerten der Nachbarspielfiguren aufgerufen.
Jede Spielfigur oder Spielerfigur hat einen ID. In dieser ID stehen X und Y Koordinaten. Um den möglichen Nachbarn einer Figur ausfindig machen zu können, muss geprüft werden ob diese einen Nachbarn hat. Es wird das Objekt-Array durchlaufen und dabei wird die ID der Figur verwendet, von der der Nachbar ausfindig gemacht werden soll. Eine Figur kann nur Links, Rechts, Oben und Unten einen Nachbarn habe. Das heißt ich musste 4 verschiedene Status Abfragen. Das Spiel ist so aufgebaut das von unten nach oben gespielt wird. Das Ziel hat als X-Array-Koordinate den Wert 0. Alle Figuren fangen unten bei dem X-Array-Koordinatewert 13 an. Das hat zur Folge, wenn wir nach oben prüfen möchten

bleibt muss sich der X-Array-Koordinatewert um 1 verringer. Wenn nach unten geprüft werden soll erhöt sich dieser Wert um 1. Soll der linke Nachbar geprüft werden so muss sich das Y-Array-Koordinatewert um 1 verringern und zum bleibt noch der rechte Nachbar bei dem sich Y-Array-Koordinatewert um 1 erhöt.

Nicht erfüllte Anforderungen

Da die Zeit mir persönlich zu knapp war, habe ich die grundlegenden Funktionen soweit ebtwickelt, dass Spiele von Usern erzeugt werden könne. Andere User diese Spiele beitreten können, das untereinander gechattet werden kann und das Spieler eines Spieles miteinandern Spielen können.

Dabei habe ich die unten aufgeführten Anforderungen in der Priorität etwas abgestuft.

- Highcore
- Verfahren zur Definition der AnfangsspielerIn
- Speicherkonzept für pausierte Spiele und die Rangliste und
- Navigationskonzept der Seiten/Ansichten

Das Spiel ist so aufgebaut, das die genannten Anforderungen implementierbar sind. Jedoch ist mir dieses nicht innerhalt der Zeit gelungen.

Klassen

Klasse „$“

Die Klasse beinhaltet die Funktionen, ein Element des gesetzten Parameters zurückzugeben. Die Funktion ist nicht zwingend erforderlich gewesen, aber erleichtert enorm den Zugriff auf Elemente. Je nach dem auf was für ein Element zugegriffen werden soll, wird zwischen „name“, „class“ und „id“ unterschieden. Bei der „id“ muss eine „#“ vor dem ID-Namen gesetzt werden, damit das Programm weiß, dass über eine „id“ auf das Element zugegriffen werden soll. Das gleich geschieht für „name“ und „class“. Bei „class“ lautet der Präfix „.“, der name brauch kein Präfix.

Diese Element wird beim Aufruf der Klasse, zurückgegeben.

Klasse „angemeldetAls“

Diese Klasse wird von der „login“ Klasse aufgerufen und erzeugt ein neues DIV. Dieses Div erhält den Inhalt des eingetragenen Nicknamen. Bei erfolgreicher Eintragung des Nicknamens, wird dieser über diese Klasse auf dem Bildschirm dargestellt.

Klasse „lobby“

Die Klasse dient zur Darstellung von Spielen. User können sich über die „login“ Klasse Anmelden, um die Lobby angezeigt zu bekommen.
Die Benutzer, die die Lobby offen habe, können Spiele erstellen, die anderen Benutzern dargestellt werden. Sobald ein Spiel von einem Benutzer erstellt wurde, können User diesem Spiel beitreten. Der einzige befugte Spieler, der dieses Spiel starten kann, ist der Anleger von diesem Spiel. Sobald sich ein User in ein erstelltes Spiel eingeloggt hat und es mindestens inkl. Anleger 2 Spieler in diesem Spiel vorhanden sind. Wird bei dem Spielanleger ein Button erstellt der ihm die Möglichkeit bietet dieses Spiel zu starten. Die maximale Anzahl von Spieler in einem Spiel ist auf 4 Spielern inkl. Anleger begrenzt.
Sobald der Spielanleger das Spiel startet, werden alle Benutzer, die sich zuvor in das Spiel eingeloggt haben, in das Spiel geladen und können mit dem Spielen beginnen.
Außerdem gehen die Chatnachrichten innerhalb der Spieler eines gestarteten Spieles nur an die Spieler im Spiel.

Die Anzahl der Spieler wird über das Objekt, welches in der Lobby angelegt wird bestimmt. Sobald das Spiel gestartet wird vom Spielersteller, steht im Objekt ein Wert mit der Anzahl der hinzugefügten Spieler.

Klasse „chat“

Die Klasse dient zur Ausgabe von Nachrichten im Chatfenster. Hierbei wird zwischen Nachricht die lokal angezeigt werden und Nachricht die über das Netzwerk ausgetauscht werden und bei allen angezeigt werden unterschieden.

Klasse „netzwerk“

Die Klasse dient dazu, Datenpakete durch das Netzwerk zu schicken. Ohne Diese Klasse ist ein Spiel zwischen mehreren Spielern nicht möglich. Alle Objekte die erzeugt werden und die durch das Netzwerk geschickt werden soll, müssen über diese Klasse übergeben werden, sonst ist ein miteinander Spielen nicht möglich. Die Objekte die bei der Klasse ankommen müssen in erster Linie in ein JSON-Objekt umgewandelt werden.

Hinzu kommt, dass die verschickten Daten ein Präfix bekommen und dieser mit dem restlichen Paket wird. Sobald Daten von Websocket kommen, werden diese dekodiert und es wird nach dem Präfix abgefragt, der am Anfang der Daten steht. Dieser Mechanismus hat zur Folge, dass Pakete ohne ein spezielles Präfix in meinem Programm nicht ausgewertet werden.

Klasse „login“

Die Klasse ermöglicht die Eingabe von einem Nicknamen. Dieser Nickname wird über eine „get“-Funktion global zugreifbar. Sobald der Nickname eingetragen wurde, bekommen alle, die die sich auf der Seite befinden eine Benachrichtigung darüber, dass sich ein neuer Benutzer eingeloggt hat.

Klasse „spielfeld“

Die Klasse ermöglicht eine Darstellung des Spielfeldes. Das Spielfeld wurde mittels eines Zweidimensionales-Array dargestellt. Anhand der Einträge in diesem Array, konnte mittels Status zwischen 0-7, 8 verschiedene dargestellt werden. 1-4 sind die Spielersteine, 5 sind die Sperrsteine, 6 sind die Spielfelder, auf denen die Spielfiguren gesetzt werden können und zu guter Letzt 7 für das Ziel.

Im gleichen Zuge habe ich ein weiteres Array angelegt, welches zum Teil identisch mit dem Array war. Der Unterschied hierbei lag daran, dass anstelle von Status, Objekte in dem Array lagen. Diese Objekte hatten weitere Objekte in sich. Objekte wie zum Beispiel, welche „id“ haben die Nachbarsteine. Das machte es einfacher den Weg zum nächsten Nachbar zu finden.

Durch die Initialisierung der Array habe ich mir gedacht, dass die „DIV’s“ dynamisch daraus erzeugt werden können. Das hat den Vorteil, dass die Änderung im Array liegt und man nicht noch unnötig den Quelltext in der HTML Datei anpassen muss.

Es wurde ein Dokumenten Fragment erzeugt und den einzelnen Elementen wurden spezielle Classen und ID‘s zugewiesen. Über die Classen wurde das Aussehen definiert und in der ID dieser Elemente standen die Koordinaten des Arrays. Man hätte das Array besser so angelegt das sich alle 90° gedreht hätte, denn derzeit ist das so aufgebaut, dass man die Y-Koordinate bestimmen muss und dann die X-Koordinate, was sehr unüblich in der Mathematik ist.

Nachdem die beiden Array erzeugt wurden und sich daraus das Spielfeld abbilden lassen konnte, kamen die Eventlistener dran.

Die Eventlistener wurden über den jeweiligen Spielfiguren gelegt, wenn dieser an der Reihe ist. Wurde gewürfelt und man hat die Maus über seine Figur gelegt, wurden aus dem Objekte-Array die Nachbarn des nächsten Steines ausgelesen, bis der Counter, die Anzahl des Würfels ergab. Der Counter wurde bei jedem Aufruf der „getNachbar“

Methode um 1 runter gezählt, damit so die endgültige Position ermittelt werden konnte und es zu keiner Endlosschleife kommt.

Schließlich wurde diesem Stein bzw. diesen Steinen eine Classe hinzugefügt, um über den Style den Over-Effekt zu erzeugen. Wurde der Spielerstein angeklickt, um diesen zu bewegen konnte schließlich auf die möglichen Positionen, mit Klick auf diese Positionen der SpielerStein verschoben werden.

Die Overeffekte oder auf die Klickeffekte habe ich so umgesetzt, dass sich in diesen „DIV’s“ Child’s erzeugt habe. Die Childs liegen dann auf den DIV’s.
Wurde die Maus nur über einen Spielstein gelegt und wieder von dem Spielerstein entfernt, wurden nur die möglichen Züge dargestellt. Beim entfernen von dem Spielerstein ohne drauf geklickt zu haben, wurde ein weiterer Eventlistener erstellt, der die Funktion zum entfernen der Classe, die den möglichen Zug darfstellt, auf die möglichen Züge aufruft.
War der nächste Nachbar der Spielsteine ein Sperrstein und der Counter war zu diesem Zeitpunkt noch nicht auf 0, wurde dieser weg als ungültig gezählt und es wurden nur diesen Steinen ein Over-Effekt zugewiesen, auf den der Zug möglich war und und kein Sperrstein auf dem Weg lag.

Klasse „stein“

Beinhaltet verschiedene Methoden der Spielsteine, dazu gehören nicht die Spielersteine.
Aus dem Objekte-Array kann man auf jedem Objekt die Methoden der „stein“ Klasse anwenden. Diese Klasse mit ihrem Methoden wird zum Teil von der „spielfeld“ Klasse aufgerufen, um somit die Nachbarn in die Objekt hinzuzufügen, wenn ein zug möglich ist, diesem Stein ein Child hinzuzufügen, wenn ein Stein an eine neue Position gesetzt wurde, den DIV’s, die nicht für diesen Zug verwendet wurden, dieses Child zu entfernen.

Klasse „spielSteine“

Die „spielSteine“ Klasse beinhaltet die oben genannten Funktionen zum hinzufügen der Eventlistener beim mouseover, mouseout und click Event. Ja nach Event werden verschiedene Funktionen aufgerufen. In dieser Methode werden auch die Childs, die den normalen Steinen hinzugefügt werden erzeugt. Diese Klasse kann ein Child auf in einem DIV erzeugen, ein clickevent, ein mousemoveevent und ein mouseoutevent verarbeiten.

Klasse „aktuellesSpiel“

Die „aktuellesSpiel“ Klasse steuert alle Vorgänge, die auf dem Spielfeld geschähen, während eines gestarteten Spieles.
Diese Klasse beinhaltet eine „init()“ Methode die dann aufgerufen wird, wenn der Spielersteller auf „Spiel Starten“ klickt.

Die Klasse beinhaltet Informationen über die Spielernummer, Spielnummer, Spieleranzahl & welcher Figur am Zuge.
Außerdem wird über eine Token-Variable definiert welcher Spieler an der Reihe ist. Diese Token wird wiederum hochgezählt sobald eine Figur von A nach B platziert wurde.
Nachdem der Token hochgezählt wurde ist der nächste Spieler an der Reihe. Wenn die maximale Anzahl der Spieler erreicht wurde, wird der token wieder auf 1 gesetzt, sodass der erste wieder an der Reihe ist. Während jedes Spielvorganges, werden Objekte über die „netzwerk“ Klasse, „game“ Objekte erzeugt, die dann zurück an die „aktuellesSpiel“ Klasse geschickt werden und so von jedem Spieler ausgewertet werden kann um den Zug der Mitspielers auf seinem Spielfeld abzubilden. Wenn ein Zug es erlaubt und es geschafft wird auf einen Sperrstein zu gelangen, werden Zwei Objekte in das Netzwerk gesendet um nicht nur den Zug vom Spielerstein abzubilden, sonder auch den umgesetzten Sperrstein.

Nach jedem Zug eines Spielers, wird abgefragt, ob eines der Spielfiguren das Ziel erreicht hat.
Das Ziel hat eine spezielle Koordinate und wenn diese von einem Spieler erreicht wird, erscheint eine Meldung bei dem Gewinner des Spieles.