Dokumentation

Projekt:

Netzwerkstar 2

Einleitung

Das vorliegende Projekt ist als online abrufbares Browser-Spiel ausgelegt. Es wurde mit Hilfe der Programmiersprache Javascript erstellt und nutzt den Hypertext Markup Language-Standard (HTML) in Version 5 zur Ausgabe. Darüber hinaus wird auf CSS-Formatierung und –Animationen zurückgegriffen.

Alle das Spiel betreffenden inhaltlichen Daten werden extern bereitgestellt. Zu diesem Zweck wurden alle Programmteile möglichst flexibel ausgelegt. Inhaltliche Änderungen erfordern deshalb keine großen Eingriffe in den Programmcode.

Konzepte

Die Spielinhalte werden extern bereitgestellt und müssen dem Programm auf zwei Arten zur Verfügung gestellt werden.

Alle optischen Inhalte liegen dazu als Bilddateien vor. Für diesen Zweck hat sich das Portable Network Graphics-Format (PNG) als am besten geeignet herausgestellt. Obwohl damit keine minimale Dateigröße erreicht wird, bietet es gegenüber dem Joint Photographics Experts Group File Interchange-Format (JPEG) die Möglichkeit, transparente Bildbereiche zu definieren. Das Projekt nutzt dennoch beide Formate, falls diese Funktion nicht benötigt wird.

Die übrigen Inhalte, wie zum Beispiel der Verlauf der Geschichte und die Dialoge der Charaktere, werden in Extensible Markup Language-Dateien (XML) hinterlegt. Hierbei ist auf die spezielle Formatierung und den Aufbau der Dateien zu achten. Dadurch lassen sich viele Funktionen und Eigenschaften der Inhalte im späteren Spiel beeinflussen.

Generell erfolgt eine Unterteilung der Handlung in einzelne Szenen. Diese beinhalten jeweils mehrere Dialoge zwischen den Figuren und weiterhin fest definierte Aufgaben, die durch den Spieler zu lösen sind.

Nach dem Aufruf des Spiels über einen Browser werden die Spielinhalte geladen. Dazu muss der Browser zwingend das Nachladen von Inhalten per Javascript und die Darstellung von HTML5 unterstützen.

Das Spiel beginnt mit einem Auswahlmenü in dem ein neues Spiel gestartet, ein vorheriger Stand geladen oder die Homepage aufgerufen werden kann. Nach dem Start oder dem Ladevorgang wird die entsprechende Szene angezeigt.

Die Darstellung erfolgt dabei zweidimensional. Durch die Anordnung der Inhalte kann jedoch ein dreidimensionaler Eindruck erweckt werden. Dabei werden insgesamt 5 Ebenen genutzt. Neben dem Hintergrund existieren jeweils zwei Vorder- und zwei Hintergrundebenen für statische und dynamische Gegenstände im Spiel. Während statische Gegenstände keine weitere Veränderung erlauben, ermöglichen dynamische Gegenstände zum Beispiel Animationen. Die Protagonisten können zwischen den Ebenen wechseln und tragen damit zur Tiefenwahrnehmung bei.

Die genannten Animationen werden durch Tilesets realisiert. Dabei handelt es sich um Standbilder, die mehrere kleine Einzelbilder einer Animation beinhalten. Durch das wiederholte Anzeigen dieser Einzelbilder entsteht der Eindruck einer Bewegung.

Dateiorganisation

Alle Spieldateien befinden sich in einem Verzeichnis. Auf der obersten Ebene sind definierenden XML-Dateien für die Spielszenen und die Dialoge abgelegt. Alle Bilder befinden sich im entsprechend benannten Ordner. Er beinhaltet die dritte XML-Datei, in der alle Bilddateien hinterlegt und mit

Zusatzinformationen versehen sind. Die Daten sind dabei in szenenspezifischen Unterordnern organisiert. Die beinhaltende HTML-Seite befindet sich im Unterordner „html“ des Wurzelverzeichnisses. Hier ist auch die CSS-Definitionsdatei gespeichert. Der Unterordner „js“ stellt alle Javascript-Dateien bereit. Diese sind noch einmal unterteilt. Im Ordner „external“ befindet sich die Bibliothek jQuery in der Version 1.8.3. Sie stellt viele verschiedene Komfortfunktionen bereit und wurde nicht im Rahmen dieses Projekts entwickelt. Der Ordner „core“ beinhaltet alle spielspezifischen Programmdateien.

Programmablauf

Nach dem Aufruf der HTML-Seite wird zuerst eine Animation angezeigt. Ihr Inhalt ist innerhalb der HTML-Datei definiert. Der Text wird unter Nutzung von CSS-Animationen dargestellt. Die entsprechenden Parameter befinden sich in der CSS-Definition. Für eine möglichst weitreichende Unterstützung verschiedener Browser wurden mehrere vorläufige CSS-Eigenschaften implementiert. Die entsprechenden Standards sind noch nicht in alle gängigen Programmversionen aufgenommen worden.

Nach einem Klick auf „Neues Spiel“ beginnt die Verarbeitung der Spieldaten mit dem Laden der XML-Dateien. Hierzu beinhaltet das Programm zwei Parser. Sie starten den Ladevorgang und die Verarbeitung, sobald die jeweilige Datei vollständig übertragen wurde.

Bilder

Der *pictureParser* verarbeitet die *bilder.xml*-Datei. Durch den Aufruf der Methode *ladeBilder* wird die Datei heruntergeladen. Mithilfe des Parameters force_load_common lässt sich ein Laden der Bilder im Ordner *allgemein* erzwingen. Dieser Ordner wird für szenenübergreifende Inhalte wie zum Beispiel die Protagonisten verwendet.

In der ersten Szene steht der Parameter deshalb auf *true*. Der Inhalt der Datei wird an *verarbeiteBilderXML* übergeben. Hier bestimmt die globale Variable *gcurrent_scene_counter* über die Filterung der Daten. Nur die für die aktuelle Szene relevanten Inhalte werden erfasst.

Alle globalen Variablen sind in der Datei *helper.js* definiert. Sie stellt den zentralen Speicher für verschiedene Klassen, Methoden und Variablen dar.

Die Filterung erfolgt mithilfe des *id*-Parameters. Er ist für jedes Bild in der *bilder.xml*-Datei definiert und identifiziert es eindeutig. Nachdem alle relevanten Bilder so erfasst wurden, wird ihre Anzahl im globalen Objekt *gBilder* gespeichert. Sie dient später der Überprüfung, ob alle Inhalte erfolgreich geladen werden konnten. Ähnliches gilt für das Flag *gpictureparser_xml_geladen*, welches den erfolgreichen Ladevorgang der XML-Datei signalisiert.

Das Laden der Bilddaten selbst erfolgt in der Methode *verarbeiteBilder*. Ihr wird die vorher erstellte Liste von Bildern übergeben. Innerhalb der Methode werden die weiteren XML-Attribute ausgelesen und jedes Bild als Instanz der Klasse *Bild* in *gBilder* abgelegt. Der Zugriff erfolgt dabei später über die genannte Bild-id. *Bild* speichert neben der *id* auch den Pfad und die Abmessungen des Bildes, wie sie in der XML-Datei hinterlegt sind. Falls das Bild animiert werden soll, wird das entsprechende Flag auf *true* gesetzt und zusätzlich eine Instanz der Klasse Animationsmerkmale mit den Attributen *fps*, *tile_anzahl*, *tile_width* und *tile_height* erzeugt. Sie beinhaltet die nötigen Angaben für die Darstellungen der Bildanimation.

Das instanziieren der Klasse *Bild* bewirkt den Start des Ladevorgangs. Gleichzeitig wird eine Funktion definiert, welche nach dem vollständigen Vorgang ausgeführt wird. Sie ruft unter anderem die Methode *waitforparser* in *helper.js* auf und erhöht den Zähler für geladenen Bilder in *gBilder*.

Dialoge

Der *dialogParser* arbeitet sehr ähnlich wie das Pendant für Bilder. Die Methode *verarbeiteDialogeXML* prüft mithilfe von *gInitialLoad*, ob es sich um den Beginn einer Szene handelt. In diesem Fall wird das globale Array *gDialogIDs* angelegt. Andernfalls wird dessen Inhalt in den Variablen *gDeprecatedDialogIDs* und *gBackupOfDialogs* gespeichert, bevor es überschrieben wird. Dies passiert kurz vor dem Ende einer Szene, wenn bereits die Inhalte der Folgenden vorgeladen werden. *gUseDeprecatedDialogues* wird dann auf *true* gesetzt und bewirkt damit, dass alle alten Dialoge der endenden Szene in *gDeprecatedDialogues* überführt und bis zum Beginn des nächsten Abschnitts von dort gelesen werden. *gDialoge*, als Behälter für die Szenentexte, wird damit frei für neue Inhalte. Diese werden anhand der aktuellen Szenennummer aus den XML-Daten gefiltert und dann, wie im *pictureParser*, in einzelnen Instanzen der Klasse *Dialog* gespeichert. Sie stellt später die Texte zur Anzeige anhand ihrer *id* bereit. Für die Verwendung in den Spielrätseln wird separat das Array *gDialogIDs* mit *DialogIDObject*-Instanzen gefüllt. Sie speichern die XML-Attribute, die über die Auswirkungen des Dialogs auf den Fortschritt im Spiel entscheiden. Dabei bestimmt *increase_quiz_step* darüber, ob dieser Dialog das Rätsel vorantreibt, *trigger_at_start* bewirkt eine Anzeige direkt nach Szenenbeginn ohne Spielertätigkeit und *invoke_scene_exception* stellt mit *argument_list* die Möglichkeit bereit, Ausnahmebehandlungen zu definieren.

Dabei handelt es sich um Funktionen, die in *sceneExceptioins.js* hinterlegt werden können. Sie dienen der Ausführung von Sonderfunktionen in speziellen Situationen und bieten so beliebige Erweiterungsmöglichkeiten für das gesamte Programm. i*nvoke_scene_exception* wirkt dabei gleichermaßen als Flag, als auch als Identifizierung der gewünschten Funktion, während *argument_list* die zugehörigen Parameter bereitstellt. Beide lassen sich mit dem Wert *#none#* deaktivieren.

Jeder Dialog der per *increase_quiz_step* entsprechend markiert wurde, erhöht den Zähler *gQuizTrueQuizSteps*. Da zwischen Rätselschritten und Fortschritt in der Szene unterschieden wird, ist dieser Punkt wichtig, denn nur wenige Dialoge treiben das Rätsel voran.

Jeder Dialog kann ein Gespräch zwischen mehreren Teilnehmern abbilden. Deshalb ist die XML-Datei in einzelne Sätze unterteilt und das *Dialog*-Objekt speichert diese in einem Array aus *Satz*-Instanzen. Neben dem Text definiert jeder Satz auch den Sprecher und dessen Repräsentation im Dialogfenster. Die *bild_id* ist dabei eine Referenz auf *gBilder*, welches das entsprechende Bild bereitstellt.

Wie zuvor die Bilder bewirkt die Instanziierung der Klasse *Dialog* den Aufruf von *waitforparser*.

Szene

Nachdem alle Inhalte geladen wurden, erfolgt die Anzeige. Dazu prüft die Methode *waitforparser* bei jedem Aufruf den Status der einzelnen Parser. Erst wenn alle Bilder und Dialoge geladen wurden, erfolgt der Aufruf von *getSceneInformation* mit der aktuellen Szenennummer und der zu ladenden XML-Datei. Dies geschieht allerdings nur wenn *gdisplay_next_scene* auf *true* steht, was bei der ersten Szene der Fall ist. Nachdem eine Szene durch *sceneParser.js* geladen wurde, wird die Variable umgeschaltet und erst nach dem Durchlaufen des Rätsels durch *quizControl.js* wieder zurückgesetzt.

getSceneInformation legt zu Beginn eine Instanz von *sceneStruct* an. Die Klasse beinhaltet später alle Objekte der Szene und speichert diese in fünf Arrays, eines für jede Bildebene. Die aktuelle Szene wird aus der Datei *szenen.xml* eingelesen. Jede Szene definiert ein Attribut *rätselschritte*. Es wird global in *gQuizsteps* gespeichert und dient der Überprüfung des Fortschritts. Sobald der Spieler hier den vorletzten Schritt erreicht, wird das Laden der nächsten Szene ausgelöst. Die Inhalte werden dann bei Erreichen des letzten Schrittes angezeigt.

Da einige Szenen im Weltraum spielen und dies Auswirkungen auf die Darstellung der Protagonisten hat, wird das Attribut *spc* ebenfalls ausgewertet. Für die betreffenden Szenen ist es in der XML-Datei auf *true* gesetzt.

Als *wegpunkt* sind in der Datei die Bildschirmkoordinaten gespeichert, die die Protagonisten auf dem Weg zu Spielobjekten ansteuern. Die Angaben für *x* und *y* stellen dabei relative Koordinaten innerhalb des Browser-Fensters dar und *zoom* bestimmt über die Ebene in der die Figur an der Position dargestellt wird. Ihre Größe wird dementsprechend angepasst. Der Parameter beeinflusst auch die Darstellung aller anderen Spielobjekte und bewirkt maßgeblich den Tiefenwahrnehmungseffekt. An diesem Punkt werden die Arrays *gWegPos* mit den Koordinaten und *gZoomsteps* mit einzelnen Zoomschritten gefüllt.

Die Objektebenen werden mithilfe der Methode *getSceneElementData* ausgelesen. Ihr werden alle zugeordneten XML-Elemente übergeben. Sie beinhalten die *id* des anzuzeigenden Bildes, die *id* des verknüpften Dialog-Objektes, die Rätsel-Flags sowie Positions- und Größenangaben. Letztere sind wiederum relative Angaben bezogen auf die Fensterabmessungen. Im Unterschied zu den Wegpunkten repräsentiert die z-Position jedoch den z-Index im Sinne einer CSS-Eigenschaft. Während Objekte ihre Position nicht ändern, wird dieser Wert für Personen dynamisch angepasst, wenn sie sich bewegen.

Zunächst wird ein temporäres *objectStruct*-Objekt erzeugt. Es speichert die genannten *id*-Attribute und alle Quizinformationen. Zusätzlich wird jedes Spielobjekt in *gImageToObjectSceneReferrer* aufgenommen. Dort werden die angezeigten Bilder mit den Dialogen verknüpft, welche ihnen durch die XML-Datei zugeordnet sind. Auf diese Weise muss nicht in den verschiedenen Datenbehältern nach den Objekten gesucht werden wenn der Spieler etwas anklickt.

Dem temporären Objekt werden noch die Rätsel-Flags hinzugefügt, bevor es für die weitere Auswertung an *getElementData* übergeben wird. Die Flags bestimmen neben der Sichtbarkeit auch darüber, ob ein Objekt das Rätsel vorantreibt, für den Spieler anklickbar ist und ob die Spielfiguren zu seiner Position gehen können. Die Information ist dabei als Array aus booleschen Werten kodiert. Jedes „f“ entspricht *false* und jedes „t“ dem Wert *true* für den Rätselschritt, der seinem Index im Array entspricht. Das Array beinhaltet dabei immer dieselbe Anzahl Einträge, die unter *reatselschritte* definiert wurde.

Innerhalb von *getElementData* wird neben der Position und Abmessung eines Objektes auch zusätzlich das optionale Attribut *laufziel* ausgelesen. Es definiert abweichende Koordinaten, die ein Charakter ansteuern kann. Ohne das Attribut wird stets die Mitte der unteren Kante eines Objektes als Ziel der Laufanimation gesetzt. Die Methode *getPersonElementData* wird für alle beschriebenen *person*-Elemente der XML-Datei aufgerufen. Sie lässt die Rätselattribute aus, funktioniert aber sonst analog zu *getSceneElementData*.

Nachdem jedes Element ausgelesen und dem entsprechenden Array innerhalb des *sceneObject* hinzugefügt wurde, kann die Methode *drawScene* mit diesem Objekt aufgerufen werden. Sie übernimmt die Darstellung der Szene. Anschließend wird *gdisplay_next_scene* auf *false* gesetzt, um ein Laden der nächsten Szene vor den zugehörigen Inhalten zu verhindern. Die Funktion *HideElementsMenu* blendet das Spielmenü aus und mithilfe von *scene1_hideHeroine* wird die Protagonistin in der ersten Szene ausgeblendet. Die Methode ist ein Beispiel für die Ausnahmebehandlungen in *sceneExceptions.js*.

sceneNeedsForcedDialog sucht im Array *gForceDialogScenes* nach Szenen deren Handlung die Anzeige eines Dialogs zu Beginn vorsieht. In diesem Fall wird durch die Methode *forceDialog* in

dialogControl.js der definierte Dialog angezeigt. Dies geschieht durch die Methode *dialog_zeichneDialog*.

Nachdem dieser Dialog, wenn nötig, angezeigt wurde, wird in *sceneParser.js* noch der *gImageToObjectSceneReferrer* auf Objekte geprüft, welche nicht zur aktuellen Szene gehören. Zur Vermeidung von Überschneidungen werden diese Objekte gefiltert, die sich beim Übergang zwischen Szenen ansammeln.

Abschließend erfolgt eine Prüfung, ob die Szene vom Spieler per Schlüsseleingabe geladen wurde. In diesem Fall setzt *advanceSceneToLastSavestate* das Spiel auf den geforderten Stand zurück. Die Methode ist in *codeGenerator.js* definiert und spielt alle nötigen Schritte zum Erreichen eines gegebenen Spielfortschritts ab. Dabei greift sie auf das *gCodegeneratorArray* zurück. Es beinhaltet für jede Szene ein Array mit Indexeinträgen entsprechend den Schlüsselwerten in *gClickEventValueArray*. Hier sind alle anklickbaren Objekte der Szene hinterlegt. Die Funktion emuliert den Klick auf diese Objekte und nutzt die vorhandene Logik zur Wiederherstellung des Spielstandes.

Das Zeichnen der Szene erfolgt pro Ebene in *drawScene*. Für jede Ebene wird *drawObjectsOfSameType* mit allen zuvor gesammelten Objekten, einem gemeinsamen Teil der ID und dem Flag *hasSingleCanvas* aufgerufen. Die Bilddaten der Objekte werden in der Methode in HTML *Canvas*-Elemente geschrieben. Diese sind unter einer ID erreichbar, welche sich unter anderem aus *sharedIdString* und dem *id*-Attribut des Bildes zusammensetzt. *hasSingleCanvas* bestimmt darüber, ob alle Objekte dieser Ebene stattdessen in ein einziges *Canvas*-Element gezeichnet werden. Dies trifft zum Beispiel für statische Hintergrundobjekte zu.

In jedem Fall für ein neues *Canvas*-Objekt erzeugt und dem HTML-Dokument hinzugefügt. Bei statischen Hintergrundobjekten nimmt dieses Element die Abmessungen des Fensters an. Anschließend werden alle zu zeichnenden Elemente nach ihrem z-Index sortiert. Für die Nutzung im Spielverlauf werden die wesentlichen Eigenschaften jedes Objekts an dieser Stelle innerhalb von *gImageStats* als Instanz von *imageStatObject* gespeichert. Dazu zählen die ID, Position und Größe sowie das separate l*aufziel*-Attribut des Objekts. Danach wird der Bildinhalt gezeichnet. Hierbei werden die relativen Angaben aus der XML-Datei durch *perc2pix* in *helper.js* in Bildschirmkoordinaten umgerechnet.

Abschließend wird die z-Index-Eigenschaft des *Canvas* gesetzt.

Der Ablauf für den Fall, dass *hasSingleCanvas* auf *false* oder nicht gesetzt wurde, ist sehr ähnlich aufgebaut. Wie bereits erwähnt erhält jedes *Canvas* eine eindeutige ID zusammengesetzt aus *sharedIdString* und *id*-Attribut des zugrundeliegenden Bildes. Danach erfolgt eine Unterscheidung zwischen der Personen- und den übrigen Ebenen.

Ebenenobjekte, die keine Person darstellen, erhalten weitere Merkmale, die ihrer ID hinzugefügt werden. Dazu zählen die *raetsel_sichtbar*- und *klickbar*-Flags aus der *szenen.xml*-Datei. Auf diese Weise sind die Informationen direkt verfügbar, wenn das *Canvas* angeklickt wird. Zusätzlich werden die *raetsel_ausloeser*- und *walkto*-Flags zusammen mit der ursprünglichen Canvas-ID und dem *id*-Attribut des zugehörigen Bildes in *gClickEventValueArray* gespeichert. Diese Struktur wird zum Beispiel beim Laden per Schlüsseleingabe und beim Klick auf ein Spielobjekt genutzt, um diese Informationen direkt verfügbar zu machen.

Jedem *Canvas* wird eine CSS-Klasse zugeordnet. Sie bestimmt die Sichtbarkeit des Objekts und den angezeigten Effekt wenn der Mauszeiger über dem Objekt schwebt. *raetsel_sichtbar* und *klickbar* entscheiden über das Hinzufügen der Klassen *quiz_hidden* oder *quiz_shown* und *clickable*. Zusätzlich

wird das *onclick*-Attribut des *Canvas* mit einem Funktionsaufruf versehen. Dadurch wird beim Klick darauf die Methode *startEventHandling* mit der ursprünglichen ID des Objektes aufgerufen.

Personen darstellende Objekte erhalten automatische die CSS-Klasse *quiz_shown* und keine weiteren *onclick*-Attribute.

Anschließend werden alle Ebenen wieder gleich behandelt. Die angegebenen Abmessungen werden in Bildschirmkoordinaten umgerechnet und unter Berücksichtigung der Skalierung angewendet. Diese wird mithilfe von *z2mult* in *helper.js* aus einem gegebenen z-Index berechnet und entspricht einem Wert innerhalb von *gZoomsteps*.

Die letzte Unterscheidung betrifft animierte Bilder. Statische Anzeigen werden direkt in das *Canvas*-Element gezeichnet. Animierte Bilder werden *pictureAnimation.js* und die Methode *starteAnimation* übergeben.

Animation

Laufanimation

Dialogausgabe

Rätselkontrolle

Eingabeverarbeitung