ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ
ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Отчет о прохождении производственной технологической (проектно-технологической) практики
на тему: «Реинжиниринг плагина "Цветовой тест Люшера"для Moodle»

Булыкина Елена Сергеевна, гр. 3530903/90302

Направление подготовки: 09.03.03 Прикладная информатика.
Место прохождения практики: СПбПУ, ИКНТ, ВШИСиСТ.
Сроки практики: с 11.06.2022 по 09.07.2022.
Руководитель практики от ФГАОУ ВО «СПбПУ»: Щербаков Николай, доцент ВШИСиСТ, к.т.н..
Консультант практики от ФГАОУ ВО «СПбПУ»: Пархоменко Владимир Андреевич, ассистент ВШИСиСТ.
Оценка: _____

Руководитель практики
от ФГАОУ ВО «СПбПУ» Н. Щербаков

Консультант практики
от ФГАОУ ВО «СПбПУ» В.А. Пархоменко

Обучающийся Е.С. Булыкина

Дата: 09.07.2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В XXI веке тема ментального здоровья стала настоящим трендом: если раньше многие предпочитали скрывать свои эмоции и умалчивать о проблемах, то сейчас всё больше и больше людей продвигают идею об «экологичном» проживании своих чувств. Несомненно, растёт уровень осознанности людей, растёт количество способов самоанализа внутреннего состояния, что позволяет вовремя обратиться к специалисту за помощью.

Актуальность исследования заключается в росте заинтересованности людей в их ментальном здоровье. На жизненном пути человеку встречаются как хорошие, так и плохие ситуации, которые могут негативно сказываться на психическом состоянии, одним из показателей которого является уровень тревожности. Тест Люшера – один из самых популярных тестов, его реализацию можно с лёгкостью найти в интернете, однако на других сервисах отсутствует аутентификация, из-за чего теряется ценность результатов. Система Moodle позволяет проводить тестирование среди студентов с подтверждёнными учетными записями.

Цель исследования — провести реинжиниринг плагина теста Люшера для версии Moodle 3.10. Для достижения цели необходимо решить следующие задачи:

A. Развернуть систему Moodle на своём компьютере.
B. Изучить возможности плагинов quiz.
C. Изучить особенности создания плагинов вопросов и поведения для Moodle.
D. Протестировать работу имеющегося плагина для теста Люшера.
E. Выявить ошибки работы плагина в новой версии и исправить его работу.

Результаты работы:

A. Обзор возможностей Moodle для реализации психологического тестирования.
B. Разработанные плагины для теста Люшера.
C. Результаты тестирования разработанного ПО.

ГЛАВА 1. ЗНАКОМСТВО С СДО MOODLE И РАЗВЁРТЫВАНИЕ СИСТЕМЫ НА КОМПЬЮТЕРЕ

Данная глава посвящена знакомству с СДО Moodle. В параграфе 1.1 приведены основная информация о Moodle и разбор структур плагинов вопроса и поведения. Параграф 1.2 посвящён установке Moodle на компьютер.

1.1. Знакомство с Moodle и плагинами

1.1.1. Общая информация о Moodle

Moodle — это обучающая платформа, предназначенная для предоставления преподавателям, администраторам и учащимся единой надежной, безопасной и интегрированной системы для создания персонализированной среды обучения.[1]

Moodle предоставляется бесплатно в виде программного обеспечения с открытым исходным кодом под лицензией GNU General Public. Любой желающий может адаптировать, расширять или модифицировать Moodle как для коммерческих, так и для некоммерческих проектов без каких-либо лицензионных сборов и воспользоваться экономичностью, гибкостью и другими преимуществами использования Moodle.[1]

Система позволяет разворачивать множество курсов, доступных определённым группам пользователей. Наполнение курса может содержать текстовые документы, презентации, видео, форумы для обсуждений. Все элементы являются подлючаемыми плагинами, код которых можно изучить в папке сервера.

Важной составляющей являются также тесты, для которых реализовано 15 различных типов вопросов. В данной работе нас интересует вопрос типа перетаскивания изображений, так как именно на основе него создавался новый тип вопроса для цветового теста Люшера.

1.1.2. Структура плагина Question type

В плагине определяются форма, её поля, которые преподаватель должен будет заполнить, а также отображение вопроса при прохождении теста. Для данных плагинов в документации Moodle можно найти репозиторий[2] с шаблоном, который состоит из следующих файлов:

- amd/...: папка, содержащая модули AMD на языке javascript для интерактивности вопроса;
- db/install.xml: файл создания таблиц, используемых данным типом вопроса;
- lang/. . . : языковые файлы с параметрами плагина;
- edit_typename_form.php: определение формы вопроса;
- lib.php: обслуживание файлов типа вопроса;
- question.php, questiontype.php: определение классов для нового типа;
- renderer.php: файл определяет отображение вопроса пользователю.

1.1.3. Структура плагина Question behavior

Представляет поведение вопроса во время теста Quiz: настройка перехода между вопросами, система оценивания, отображение результатов. Плагин поведения представляет собой папку с файлами, отвечающими за работу плагина:

- behaviour.php: содержит описания класса поведения вопроса;
- renderer.php: содержит определение класса qbehaviour_renderer;
- lang/. . . : языковые файлы с параметрами плагина;
- behaviourtype.php: содержит определение класса qbehaviour_type.[5]

1.2. Установка Moodle на локальный компьютер

Для системы с небольшим количеством пользователей установка Moodle производится с помощью специального архива, содержащего файл типа «.exe», который автоматически установит Apache и MySQL. Поскольку на локальном компьютере Moodle нам нужен только для тестирования разработанных плагинов, мы воспользовались именно таким установочным пакетом.

После запуска сервера необходимо перейти в браузер на стандартную страницу localhost для дальнейшей настройки Moodle, в ходе которой нужно настроить базу данных, создать учётную запись администратора, задать имя сайта.

Данный способ установки позволяет также одновременно развёртывать несколько версий на одном компьютере, что очень важно для решения проблемы реинжиниринга. Однако, для перехода между версиями необходимо останавливать один сервер и запускать другой.

ГЛАВА 2. ИССЛЕДОВАНИЕ РАБОТЫ ГОТОВЫХ ПЛАГИНОВ

Глава посвящена тестированию плагина и поиску ошибок

В параграфе 2.1 приведён процесс и результаты тестирования плагинов в Moodle 3.5. Параграф 2.2 содержит результат тестирования плагина вопроса в Moodle 3.10.

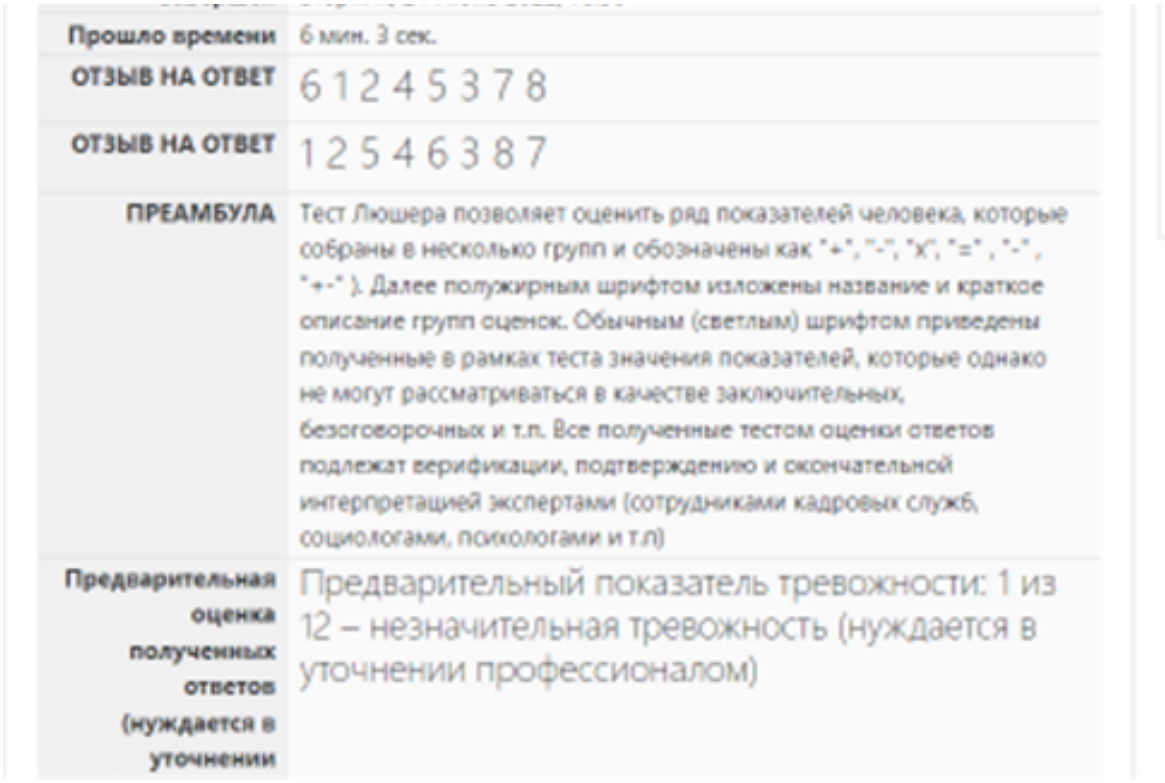
2.1. Исследование работы плагинов в Moodle 3.5

Для наглядного примера работы теста Люшера в системе Moodle была установлена версия 3.5, для которой изначально создавались плагины. В ходе первой установки ddlusher[3] система вернула ошибку в инициализации базы данных в файле install.xml: в таблице lusherr содержалось 2 поля с автоматической генерацией значения. После анализа кода стало ясно, что одному из полей не требуется автоматическая генерация при добавлении записи в таблицу, поэтому свойство sequence было снято с поля. Также было исправлено имя таблицы: с mdl_lusherr на lusherr, так как при установке плагина система самостоятельно добавляет префикс к названию. После исправлений плагин был успешно установлен.

Плагин lusherr[4] установился без ошибок с первого раза.

Для проверки функционирования был создан курс, тест и вопросы. Тест успешно отработал и вывел результаты: карточки при перемещении на фон исчезали, между вопросами запускался таймер.

На рис.2.1 приведён скриншот результатов теста.



Прошло времени	6 мин. 3 сек.
ОТЗЫВ НА ОТВЕТ	6 1 2 4 5 3 7 8
ОТЗЫВ НА ОТВЕТ	1 2 5 4 6 3 8 7
ПРЕАМБУЛА	Тест Люшера позволяет оценить ряд показателей человека, которые собраны в несколько групп и обозначены как "+", "-", "X", "=" , "-" , "+-"). Далее полужирным шрифтом изложены название и краткое описание групп оценок. Обычным (светлым) шрифтом приведены полученные в рамках теста значения показателей, которые однако не могут рассматриваться в качестве заключительных, безоговорочных и т.п. Все полученные тестом оценки ответов подлежат верификации, подтверждению и окончательной интерпретацией экспертами (сотрудниками кадровых служб, социологами, психологами и т.п)
Предварительная оценка полученных ответов (нуждается в уточнении	Предварительный показатель тревожности: 1 из 12 – незначительная тревожность (нуждается в уточнении профессионалом)

Рис.2.1. Результаты теста в версии Moodle 3.5

2.2. Тестирование плагинов в Moodle 3.10

Для дальнейшей проверки на компьютере была дополнительно развернута версия 3.10. Установка плагинов прошла без ошибок, однако при попытке создать вопрос в консоли появилась ошибка в файле с Javascript кодом, из-за которой стало невозможно редактировать поля формы.

На рис.2.2 приведён скриншот результатов теста.

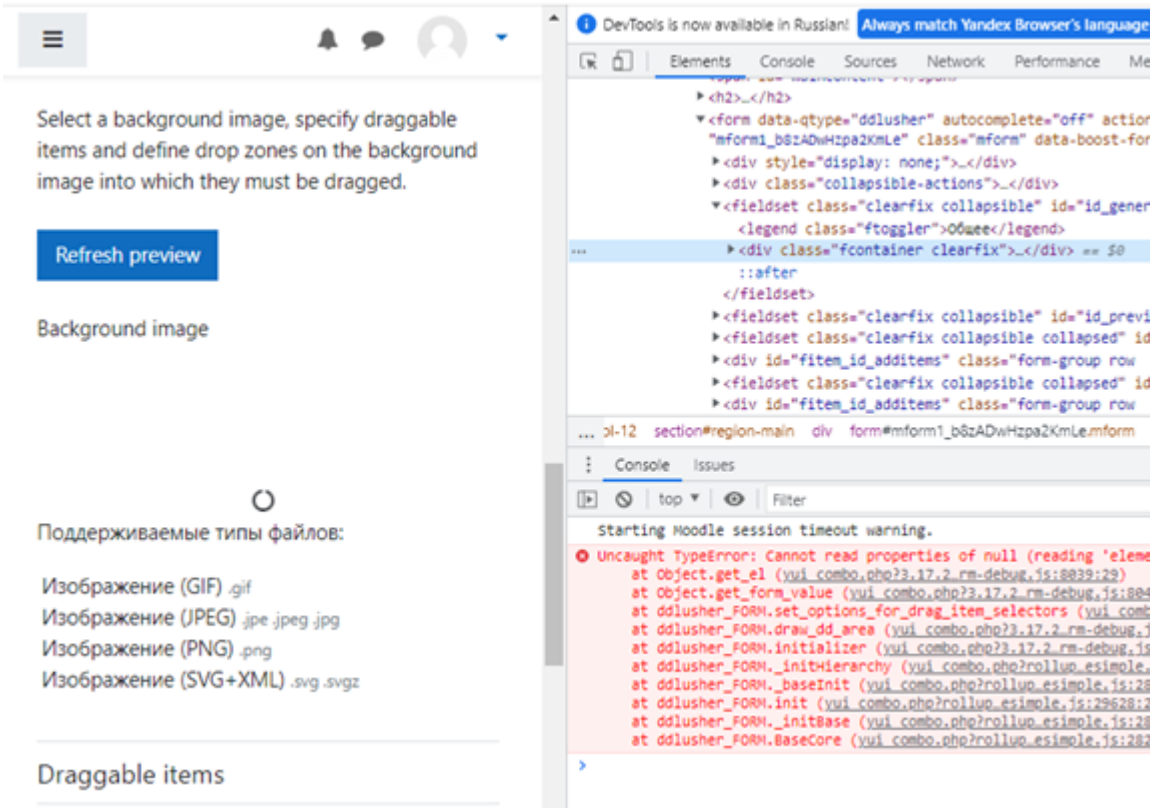


Рис.2.2. Ошибка при попытке создания вопроса в Moodle 3.10

Плагин поведения протестировать на данном этапе невозможно, так как он зависит от вопроса.

ГЛАВА 3. РЕИНЖИНИРИНГ ПЛАГИНОВ

Глава посвящена процессу реинжиниринга плагинов для теста Люшера. Параграф 3.1 посвящен реинжинирингу плагина вопроса ddlusher. В параграфе 3.2 описывается процесс анализа и обновления плагина поведения lusherr.

3.1. Реинжиринг плагина вопроса

На официальном сайте Moodle упоминается, что начиная с Moodle 2.9 осуществляется переход от модулей YUI к модулям AMD, так как команда YUI прекратила новые разработки в библиотеки.[6]

В предыдущей главе было установлено, что ошибка возникает именно в коде модуля YUI, поэтому было принято решение проверить текущий код плагина вопроса перетаскивания изображений на сервере. Как оказалось, в версии 3.10 данный тип вопроса действительно уже использует AMD.

Далее были проанализированы коды плагина перестаскивания на сервере и плагина вопроса для теста Люшера[3]. В ходе анализа выяснилось, что функции в файлах «.php» не изменились, поэтому данные файлы полностью взяты из плагина ddlusher[3]. Однако функции в модулях AMD и YUI уже существенно отличаются.

Было установлено, что с помощью javascript в ddlusher[3] осуществлялось исчезание карточек после перетаскивания и обновление поля countDrop, которое подсчитывает количество уже выбранных и перенесённых цветов.

За основу был взят код для плагина вопроса с перетаскиванием изображений из папки сервера. В ходе анализа кода модуля в файле question.js была найдена функция dragEnd, которая реагирует на конец перетаскивания. В данную функцию был вставлен код, который скрывает карточку на фоновом изображении и обновляет поле countDrop. Изменения можно найти в строках] 347-351 приложения П1.2.

Также в файле form.js было скрыто поле, отвечающее за неоднократное перетаскивание элемента на фоновое изображение, и кнопки для добавления новых элементов перетаскивания(строки 59-63 приложения П1.1).

Полный код модуля приведён в приложении 1.

3.2. Реинжиниринг плагина поведения

В первую очередь код плагина lusherr[4] был проанализирован. Из файла nobutton.js был исключен код, который скрывает кнопку перехода на предыдущий вопрос, так как предотвращение свободного перехода по вопросам можно реализовать в настройках теста.

Также был изменён текст преамбулы, отображающейся в результатах, так как подробная расшифровка теста была исключена автором.

После реинжиниринга плагина вопроса появилась возможность протестировать работу плагина поведения. Переход между вопросами и обработка результатов прошла успешно, в дополнительных доработках данный плагин не нуждается.

ГЛАВА 4. ТЕСТИРОВАНИЕ РАБОТЫ ПЛАГИНОВ В ТЕСТЕ

В главе приведены результаты работы плагинов.

4.1. Настройка Moodle

Для тестирования работы плагинов в Moodle необходимо выполнить простую установку, загрузив два архива через панель администрирования.

На рис.4.1 приведён скриншот загрузки zip-архива.

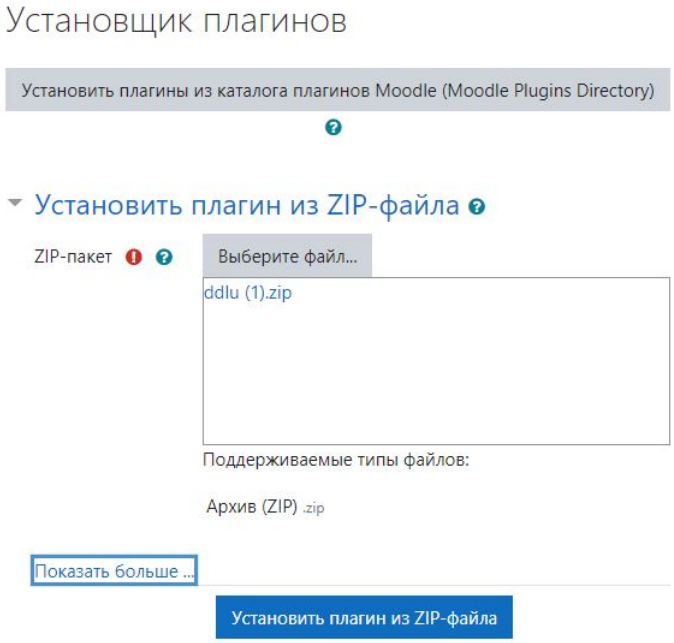


Рис.4.1. Загрузка плагинов в Moodle

Далее нужно создать тест, задать ему имя. Во вкладке «Расположение» нужно выбрать метод навигации «Последовательный», в свойствах вопроса настроить режим поведения как пользовательский LusherTest, который мы загрузили ранее. Далее перейдём на вкладку «Настройки просмотра», где уберём все галочки, связанные с баллами и правильными ответами, так система оценивания Moodle не подходит для нашего теста.

На рис.4.2 приведён скриншот с настройками расположения и свойств вопросов. На рис.4.3 приведены настройки просмотра результатов.

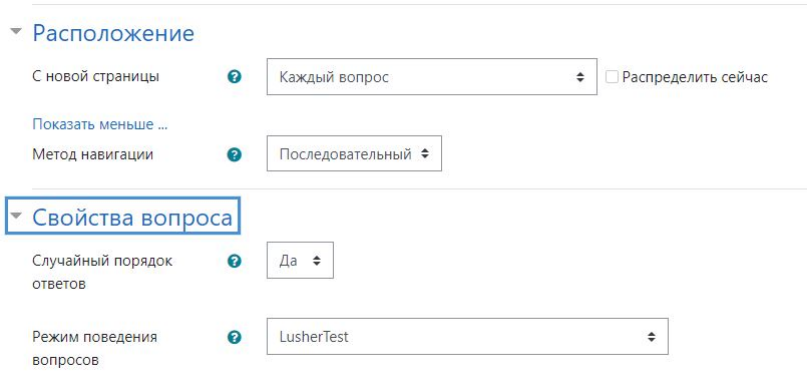


Рис.4.2. Настройки пунктов «Расположение» и «Свойства вопроса»

Рис.4.3. Настройки просмотра

В тесте было создано 2 вопроса загруженного нами типа.

4.2. Тестирование работы

После запуска теста на экране появились все карточки и кнопка продолжения, которая не работает, если цвета не перенесены на фоновое изображение. Панель навигации видна, но не работает, как и следовало ожидать. Карточки при перетаскивании исчезали. Исчезновение карточек приведено на рис.4.4.

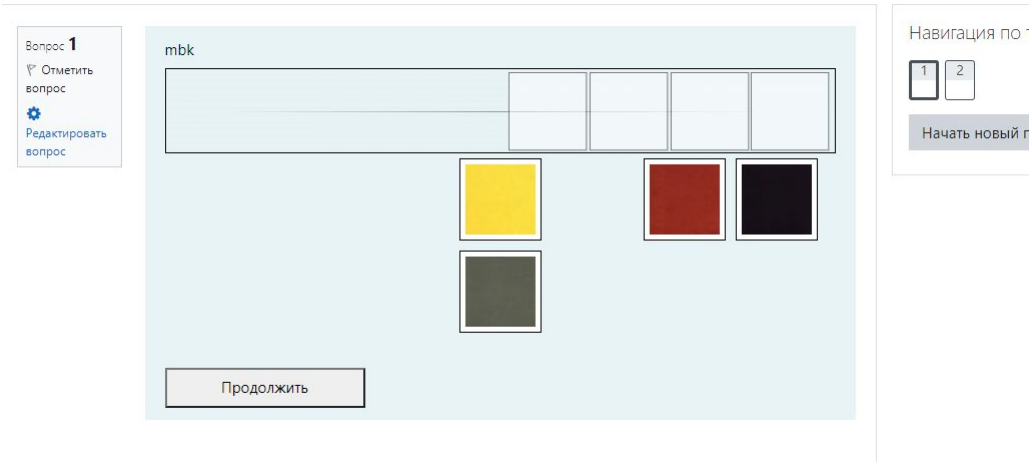


Рис.4.4. Исчезновение карточек

Между вопросами запустился таймер, по истечении времени которого стала доступна кнопка перехода ко второму вопросу. На рис.4.5 отображена страница с таймером.

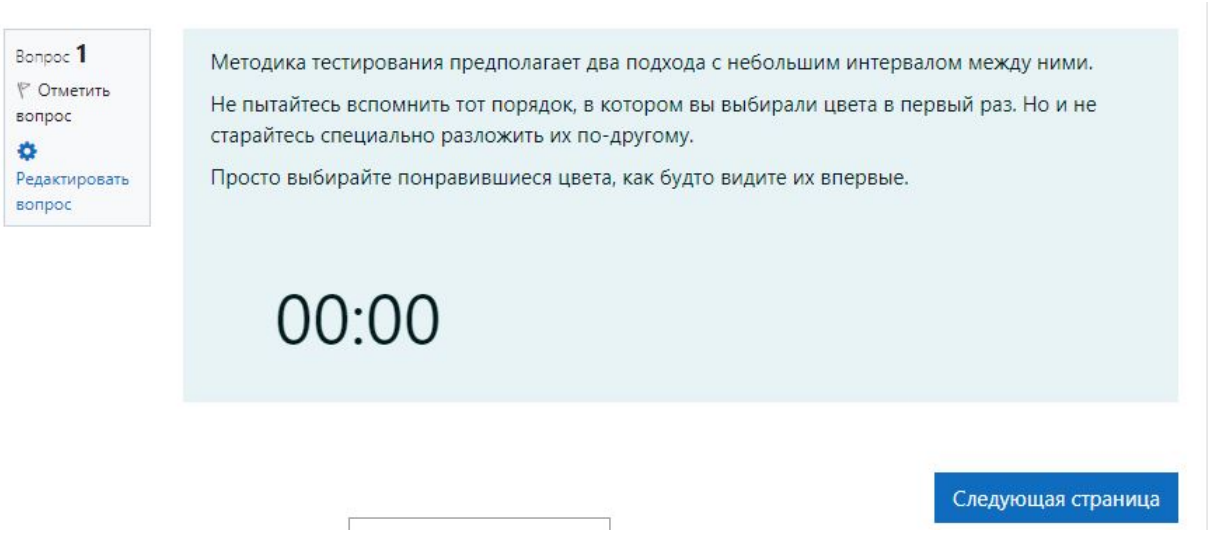


Рис.4.5. Таймер между вопросами

Тест успешно завершился и показал предварительную оценку тревожности. На рис.4.6 и рис.4.7 можно увидеть страницу отображения результатов теста.

Тест начат	понедельник, 4 Июль 2022, 18:20
Состояние	Завершено
Завершен	понедельник, 4 Июль 2022, 18:24
Прошло времени	4 мин. 1 сек.
ОТЗЫВ НА ОТВЕТ	1 2 3 5 4 8 6 7
ОТЗЫВ НА ОТВЕТ	1 2 5 4 7 6 8 3
ПРЕАМБУЛА	Тест Люшера позволяет оценить ряд показателей человека, которые собраны в несколько групп и обозначены как "+", "-", "x", "=" , "-" , "+-"). Полученная тестом оценка подлежит верификации, подтверждению и окончательной интерпретации экспертами (сотрудниками кадровых служб, социологами, психологами и т.п)
Предварительная оценка полученных ответов (нуждается в уточнении профессионалом)	Предварительный показатель тревожности: 3 из 12 – эмоциональная напряженность (нуждается в уточнении профессионалом)

Рис.4.6. Оценка тревожности

Рис.4.7. Отображение карточек в результатах

ЗАКЛЮЧЕНИЕ

В результате пройденной технологической практики была изучена система Moodle, структура плагинов типа вопроса и поведения. Также система Moodle была развёрнута на локальном компьютере.

Существующие плагины для теста Люшера были протестированы в версиях Moodle 3.5 и 3.10. Выявленные ошибки были исправлены путём обновления модуля языка javascript.

Обновлённые плагины были протестированы в Moodle 3.10 на локальном компьютере. Работа теста Люшера прошла успешно, плагины функционируют без ошибок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. About Moodle. — URL: https://docs.moodle.org/400/en/About_Moodle (visited on 20.06.2022).

2. moodle-qtype-template. — URL: https://github.com/marcusgreen/moodle-qtype_TEMPLATE/ (visited on 15.06.2022).

3. Plugin-ddlusher. — URL: https://github.com/R-D4S/Plugins-for-Psychological-tests/tree/main/ddlusher (visited on 14.06.2022).

4. plugin-lusherr. — URL: https://github.com/R-D4S/Plugins-for-Psychological-tests/tree/main/lusherr (visited on 14.06.2022).

5. Question behaviours. — URL: https://docs.moodle.org/dev/Question_behaviours (visited on 17.06.2022).

6. YUI. — URL: https://docs.moodle.org/dev/YUI (visited on 20.06.2022).

Приложение 1

Код AMD модуля плагина вопроса

П1.1. form.js

```
/*
* JavaScript to allow dragging options to slots
* (using mouse down or touch) or tab through slots using keyboard.
*
* @module      qtype_ddlu/form
* @copyright   2018 The Open University
* @license     http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
*/
define(['jquery', 'core/dragdrop'], function($, dragDrop) {

  "use strict";

  /**
  * Singleton object to handle progressive enhancement of the
  * drag-drop onto image question editing form.
  * @type {Object}
  */
  var dragDropToImageForm = {
    /**
    * @var {Object} maxBgImageSize Properties width and height.
    * @private
    */
    maxBgImageSize: null,

    /**
    * @var {Object} maxDragImageSize with properties width and height.
    * @private
    */
    maxDragImageSize: null,

    /**
    * @property {object} fp for interacting with the file pickers.
    * @private
    */
    fp: null, // Object containing functions associated with the file picker.
```

```
        /**
        * Initialise the form javascript features.
        *
        * @method
        */
        init: function() {
          dragDropToImageForm.fp = dragDropToImageForm.filePickers();

          $('#id_previewareaheader').append(
          '<div class="ddarea que ddlu">' +
          '  <div class="droparea">' +
          '    <img class="dropbackground" />' +
          '    <div class="dropzones"></div>' +
          '  </div>' +
          '  <div class="dragitems"></div>' +
          '</div>');

          dragDropToImageForm.updateVisibilityOfFilePickers();
          dragDropToImageForm.setOptionsForDragItemSelectors();
          dragDropToImageForm.setupEventHandlers();
          dragDropToImageForm.waitForFilePickerToInitialise();
          //скрываем кнопки добавления
          var ad = document.getElementsByName("additems");
          ad.item(0).setAttribute("hidden","true");
          var ad = document.getElementsByName("adddropzone");
          ad.item(0).setAttribute("hidden","true");
        },

        /**
        * Waits for the file-pickers to be sufficiently ready before
           initialising the preview.
        */
        waitForFilePickerToInitialise: function() {
          if (dragDropToImageForm.fp.file('bgimage').href === null) {
            // It would be better to use an onload or onchange event
               rather than this timeout.
            // Unfortunately attempts to do this early are overwritten
               by filepicker during its loading.
            setTimeout(dragDropToImageForm.waitForFilePickerToInitialise
               , 1000);
            return;
          }
          M.util.js_pending('dragDropToImageForm');
```

```
        // From now on, when a new file gets loaded into the
           filepicker, update the preview.
        // This is not in the setupEventHandlers section as it needs
           to be delayed until
        // after filepicker's javascript has finished.
        $('form.mform[data-qtype="ddlu"]').on('change', '.
           filepickerhidden', function() {
          M.util.js_pending('dragDropToImageForm');
          dragDropToImageForm.loadPreviewImage();
        });

        dragDropToImageForm.loadPreviewImage();
      },

      /**
      * Loads the preview background image.
      */
      loadPreviewImage: function() {
        $('fieldset#id_previewareaheader .dropbackground')
        .one('load', dragDropToImageForm.afterPreviewImageLoaded)
        .attr('src', dragDropToImageForm.fp.file('bgimage').href);
      },

      /**
      * After the background image is loaded, continue setting up the
         preview.
      */
      afterPreviewImageLoaded: function() {
        dragDropToImageForm.createDropZones();
        M.util.js_complete('dragDropToImageForm');
      },

      /**
      * Create, or recreate all the drop zones.
      */
      createDropZones: function() {
        var dropZoneHolder = $('.dropzones');
        dropZoneHolder.empty();

        var bgimageurl = dragDropToImageForm.fp.file('bgimage').href;
        if (bgimageurl === null) {
          return; // There is not currently a valid preview to update.
        }

        var numDrops = dragDropToImageForm.form.getFormValue('
           nodropzone', []);
```

```
    for (var dropNo = 0; dropNo < numDrops; dropNo++) {
      var dragNo = dragDropToImageForm.form.getFormValue('drops',
      [dropNo, 'choice']);
      if (dragNo === '0') {
        continue;
      }
      dragNo = dragNo - 1;
      var group = dragDropToImageForm.form.getFormValue('drags',
      [dragNo, 'draggroup']),
      label = dragDropToImageForm.form.getFormValue('draglabel',
      [dragNo]);
      if ('image' === dragDropToImageForm.form.getFormValue('drags
          ',
      [dragNo, 'dragitemtype'])) {
        var imgUrl = dragDropToImageForm.fp.file('dragitem[' +
            dragNo + ']').href;
        if (imgUrl === null) {
          continue;
        }
        // Althoug these are previews of drops, we also add the
            class name 'drag',
        dropZoneHolder.append('<img class="droppreview group' +
            group + ' drop' + dropNo +
        '" src="' + imgUrl + '" alt="' + label + '" data-drop-no
            ="' + dropNo + '">');

      } else if (label !== '') {
        dropZoneHolder.append('<div class="droppreview group' +
            group + ' drop' + dropNo +
        '"  data-drop-no="' + dropNo + '">' + label + '</div>');
      }
    }

    dragDropToImageForm.waitForAllDropImagesToBeLoaded();
  },

  /**
  * This polls until all the drop-zone images have loaded, and
     then calls updateDropZones().
  */
  waitForAllDropImagesToBeLoaded: function() {
    var notYetLoadedImages = $('.dropzones img').not(function(i,
       imgNode) {
      return dragDropToImageForm.imageIsLoaded(imgNode);
    });
```

```
    if (notYetLoadedImages.length > 0) {
      setTimeout(function() {
        dragDropToImageForm.waitForAllDropImagesToBeLoaded();
      }, 100);
      return;
    }

    dragDropToImageForm.updateDropZones();
  },

  /**
  * Check if an image has loaded without errors.
  *
  * @param {HTMLImageElement} imgElement an image.
  * @returns {boolean} true if this image has loaded without
     errors.
  */
  imageIsLoaded: function(imgElement) {
    return imgElement.complete && imgElement.naturalHeight !== 0;
  },

  /**
  * Set the size and position of all the drop zones.
  */
  updateDropZones: function() {
    var bgimageurl = dragDropToImageForm.fp.file('bgimage').href;
    if (bgimageurl === null) {
      return; // There is not currently a valid preview to update.
    }

    var dropBackgroundPosition = $('fieldset#id_previewareaheader
       .dropbackground').offset(),
    numDrops = dragDropToImageForm.form.getFormValue('nodropzone',
        []);

    // Move each drop to the right position and update the text.
    for (var dropNo = 0; dropNo < numDrops; dropNo++) {
      var drop = $('.dropzones .drop' + dropNo);
      if (drop.length === 0) {
        continue;
      }
      var dragNo = dragDropToImageForm.form.getFormValue('drops',
      [dropNo, 'choice']) - 1;

      drop.offset({
        left: dropBackgroundPosition.left +
```

```
            parseInt(dragDropToImageForm.form.getFormValue('drops',
            [dropNo, 'xleft'])),
            top: dropBackgroundPosition.top +
            parseInt(dragDropToImageForm.form.getFormValue('drops',
            [dropNo, 'ytop']))
          });

          var label = dragDropToImageForm.form.getFormValue('draglabel
              ', [dragNo]);
          if (drop.is('img')) {
            drop.attr('alt', label);
          } else {
            drop.html(label);
          }
        }

        // Resize them to the same size.
        $('.dropzones .droppreview').css('padding', '0');
        var numGroups = $('.draggroup select').first().find('option').
            length;
        for (var group = 1; group <= numGroups; group++) {
          dragDropToImageForm.resizeAllDragsAndDropsInGroup(group);
        }
      },

      /**
      * In a given group, set all the drags and drops to be the same
          size.
      *
      * @param {int} group the group number.
      */
      resizeAllDragsAndDropsInGroup: function(group) {
        var drops = $('.dropzones .droppreview.group' + group),
        maxWidth = 0,
        maxHeight = 0;

        // Find the maximum size of any drag in this groups.
        drops.each(function(i, drop) {
          maxWidth = Math.max(maxWidth, Math.ceil(drop.offsetWidth));
          maxHeight = Math.max(maxHeight, Math.ceil(drop.offsetHeight)
              );
        });

        // The size we will want to set is a bit bigger than this.
        maxWidth += 10;
        maxHeight += 10;
```

```
        // Set each drag home to that size.
        drops.each(function(i, drop) {
          var left = Math.round((maxWidth - drop.offsetWidth) / 2),
          top = Math.floor((maxHeight - drop.offsetHeight) / 2);
          // Set top and left padding so the item is centred.
          $(drop).css({
            'padding-left': left + 'px',
            'padding-right': (maxWidth - drop.offsetWidth - left) + '
              px',
            'padding-top': top + 'px',
            'padding-bottom': (maxHeight - drop.offsetHeight - top) +
              'px'
          });
        });
      },

      /**
      * Events linked to form actions.
      */
      setupEventHandlers: function() {
        // Changes to settings in the draggable items section.
        $('fieldset#id_draggableitemheader')
        .on('change input', 'input, select', function(e) {
          var input = $(e.target).closest('select, input');
          if (input.hasClass('dragitemtype')) {
            dragDropToImageForm.updateVisibilityOfFilePickers();
          }

          dragDropToImageForm.setOptionsForDragItemSelectors();

          if (input.is('.dragitemtype, .draggroup')) {
            dragDropToImageForm.createDropZones();
          } else if (input.is('.draglabel')) {
            dragDropToImageForm.updateDropZones();
          }
        });

        // Changes to Drop zones section: left, top and drag item.
        $('fieldset#id_dropzoneheader').on('change input', 'input,
          select', function(e) {
          var input = $(e.target).closest('select, input');
          if (input.is('select')) {
            dragDropToImageForm.createDropZones();
          } else {
            dragDropToImageForm.updateDropZones();
```

```
285           }
            });

            // Moving drop zones in the preview.
            $('fieldset#id_previewareaheader').on('mousedown touchstart',
                '.droppreview', function(e) {
              dragDropToImageForm.dragStart(e);
            });

            $(window).on('resize', function() {
              dragDropToImageForm.updateDropZones();
            });
          },

          /**
          * Update all the drag item filepickers, so they are only shown
             for
          */
          updateVisibilityOfFilePickers: function() {
            var numDrags = dragDropToImageForm.form.getFormValue('noitems
                ', []);
            for (var dragNo = 0; dragNo < numDrags; dragNo++) {
              var picker = $('input#id_dragitem_' + dragNo).closest('.
                  fitem_ffilepicker');
              if ('image' === dragDropToImageForm.form.getFormValue('drags
                  ',
              [dragNo, 'dragitemtype'])) {
                picker.show();
              } else {
                picker.hide();
              }
            }
          },


          setOptionsForDragItemSelectors: function() {
            var dragItemOptions = {'0': ''},
            numDrags = dragDropToImageForm.form.getFormValue('noitems',
                []),
            numDrops = dragDropToImageForm.form.getFormValue('nodropzone',
                 []);

            // Work out the list of options.
            for (var dragNo = 0; dragNo < numDrags; dragNo++) {
              var label = dragDropToImageForm.form.getFormValue('draglabel
                  ', [dragNo]);
```

```
        var file =
        dragDropToImageForm.fp.file(dragDropToImageForm.form.
            toNameWithIndex('dragitem',
325         [dragNo]));
        if ('image' === dragDropToImageForm.form.getFormValue('drags
            ',
        [dragNo, 'dragitemtype']) && file.name !== null) {
          dragItemOptions[dragNo + 1] = (dragNo + 1) + '. ' + label
              + ' (' + file.name + ')';
        } else if (label !== '') {
330       dragItemOptions[dragNo + 1] = (dragNo + 1) + '. ' + label;
        }
      }

      // Initialise each select.
335   for (var dropNo = 0; dropNo < numDrops; dropNo++) {
        var selector = $('#id_drops_' + dropNo + '_choice');

        var selectedvalue = selector.val();
        selector.find('option').remove();
340     for (var value in dragItemOptions) {
          if (!dragItemOptions.hasOwnProperty(value)) {
            continue;
          }
          selector.append('<option value="' + value + '">'
345       + dragItemOptions[value] + '</option>');
          var optionnode = selector.find('option[value="' + value +
              '"]');
          if (parseInt(value) === parseInt(selectedvalue)) {
            optionnode.attr('selected', true);
          } else if (dragDropToImageForm.isItemUsed(parseInt(value))
              ) {
350         optionnode.attr('disabled', true);
          }
        }
      }
    },
355

    /**
    * Checks if the specified drag option is already used somewhere.
    *
    * @param {Number} value of the drag item to check
360 * @return {Boolean} true if item is allocated to dropzone
    */
    isItemUsed: function(value) {
      if (value === 0) {
```

```
        return false; // None option can always be selected.
      }

      return $('fieldset#id_dropzoneheader select').filter(function(
          i, selectNode) {
        return parseInt($(selectNode).val()) === value;
      }).length !== 0;
    },

    /**
    * Handles when a dropzone in dragged in the preview.
    * @param {Object} e Event object
    */
    dragStart: function(e) {
      var drop = $(e.target).closest('.droppreview');

      var info = dragDrop.prepare(e);
      if (!info.start) {
        return;
      }

      dragDrop.start(e, drop, function(x, y, drop) {
        dragDropToImageForm.dragMove(drop);
      }, function() {
        dragDropToImageForm.dragEnd();
      });
    },

    /**
    * Handles update while a drop is being dragged.
    *
    * @param {jQuery} drop the drop preview being moved.
    */
    dragMove: function(drop) {
      var backgroundImage = $('fieldset#id_previewareaheader .
          dropbackground'),
      backgroundPosition = backgroundImage.offset(),
      dropNo = drop.data('dropNo'),
      dropPosition = drop.offset(),
      left = Math.round(dropPosition.left - backgroundPosition.left)
          ,
      top = Math.round(dropPosition.top - backgroundPosition.top);

      // Constrain coordinates to be inside the background.
      left = Math.round(Math.max(0, Math.min(left, backgroundImage.
          outerWidth()
```

```
        - drop.outerWidth())));
        top = Math.round(Math.max(0, Math.min(top, backgroundImage.
            outerHeight()
        - drop.outerHeight())));

410     // Update the form.
        dragDropToImageForm.form.setFormValue('drops', [dropNo, 'xleft
            '], left);
        dragDropToImageForm.form.setFormValue('drops', [dropNo, 'ytop
            '], top);
      },

415   /**
      * Handles when the drag ends.
      */
      dragEnd: function() {
        // Redraw, in case the position was constrained.
420     dragDropToImageForm.updateDropZones();
      },

      /**
      * Low level operations on form.
425   */
      form: {
        toNameWithIndex: function(name, indexes) {
          var indexString = name;
          for (var i = 0; i < indexes.length; i++) {
430         indexString = indexString + '[' + indexes[i] + ']';
          }
          return indexString;
        },

435     getEl: function(name, indexes) {
          var form = $('form.mform[data-qtype="ddlu"]')[0];
          return form.elements[this.toNameWithIndex(name, indexes)];
        },

440     /**
        * Helper to get the value of a form elements with name like "
            drops[0][xleft]".
        *
        * @param {String} name the base name, e.g. 'drops'.
        * @param {String[]} indexes the indexes, e.g. ['0', 'xleft'].
445     * @return {String} the value of that field.
        */
        getFormValue: function(name, indexes) {
```

```
        var el = this.getEl(name, indexes);
        if (!el.type) {
          el = el[el.length - 1];
        }
        if (el.type === 'checkbox') {
          return el.checked;
        } else {
          return el.value;
        }
      },

      /**
      * Helper to get the value of a form elements with name like "
         drops[0][xleft]".
      *
      * @param {String} name the base name, e.g. 'drops'.
      * @param {String[]} indexes the indexes, e.g. ['0', 'xleft'].
      * @param {String|Number} value the value to set.
      */
      setFormValue: function(name, indexes, value) {
        var el = this.getEl(name, indexes);
        if (el.type === 'checkbox') {
          el.checked = value;
        } else {
          el.value = value;
        }
      }
    },

    /**
    * Utility to get the file name and url from the filepicker.
    * @returns {Object} object containing functions {file, name}
    */
    filePickers: function() {
      var draftItemIdsToName;
      var nameToParentNode;

      if (draftItemIdsToName === undefined) {
        draftItemIdsToName = {};
        nameToParentNode = {};
        var fp = $('form.mform[data-qtype="ddlu"] input.
           filepickerhidden');
        fp.each(function(index, filepicker) {
          draftItemIdsToName[filepicker.value] = filepicker.name;
          nameToParentNode[filepicker.name] = filepicker.parentNode;
        });
```

```
        }

        return {
          file: function(name) {
            var parentNode = $(nameToParentNode[name]);
            var fileAnchor = parentNode.find('div.filepicker-filelist
                a');
            if (fileAnchor.length) {
              return {href: fileAnchor.get(0).href, name: fileAnchor.
                  get(0).innerHTML};
            } else {
              return {href: null, name: null};
            }
          },

          name: function(draftitemid) {
            return draftItemIdsToName[draftitemid];
          }
        };
      }
    };

    return {
      init: dragDropToImageForm.init
    };
  });
```

II1.2. question.js

```
/*
* JavaScript to allow dragging options to slots (using mouse down or
    touch) or tab through slots using keyboard.
*
* @module      qtype_ddlu/question
* @copyright   2018 The Open University
* @license     http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or
   later
*/
define(['jquery', 'core/dragdrop', 'core/key_codes'], function($,
   dragDrop, keys) {

  "use strict";

  /**
```

```
    * Initialise one drag-drop onto image question.
    *
    * @param {String} containerId id of the outer div for this
        question.
    * @param {boolean} readOnly whether the question is being
        displayed read-only.
    * @param {Array} places Information about the drop places.
    * @constructor
    */
    function DragDropOntoImageQuestion(containerId, readOnly, places)
        {
      this.containerId = containerId;
      M.util.js_pending('qtype_ddlu-init-' + this.containerId);
      this.places = places;
      this.allImagesLoaded = false;
      this.imageLoadingTimeoutId = null;
      this.isPrinting = false;
      if (readOnly) {
        this.getRoot().addClass('qtype_ddlu-readonly');
      }

      var thisQ = this;
      this.getNotYetLoadedImages().one('load', function() {
        thisQ.waitForAllImagesToBeLoaded();
      });
      this.waitForAllImagesToBeLoaded();
    }

    /**
    * Waits until all images are loaded before calling setupQuestion()
        .
    *
    * This function is called from the onLoad of each image, and also
        polls with
    * a time-out, because image on-loads are allegedly unreliable.
    */
    DragDropOntoImageQuestion.prototype.waitForAllImagesToBeLoaded =
        function() {
      var thisQ = this;

      // This method may get called multiple times (via image on-loads
          or timeouts.
      // If we are already done, don't do it again.
      if (this.allImagesLoaded) {
        return;
      }
```

```
    // Clear any current timeout, if set.
    if (this.imageLoadingTimeoutId !== null) {
      clearTimeout(this.imageLoadingTimeoutId);
    }

    // If we have not yet loaded all images, set a timeout to
    // call ourselves again, since apparently images on-load
    // events are flakey.
    if (this.getNotYetLoadedImages().length > 0) {
      this.imageLoadingTimeoutId = setTimeout(function() {
        thisQ.waitForAllImagesToBeLoaded();
      }, 100);
      return;
    }

    // We now have all images. Carry on, but only after giving the
       layout a chance to settle down.
    this.allImagesLoaded = true;
    thisQ.setupQuestion();
  };

  /**
  * Get any of the images in the drag-drop area that are not yet
     fully loaded.
  *
  * @returns {jQuery} those images.
  */
  DragDropOntoImageQuestion.prototype.getNotYetLoadedImages =
     function() {
    var thisQ = this;
    return this.getRoot().find('.ddarea img').not(function(i,
       imgNode) {
      return thisQ.imageIsLoaded(imgNode);
    });
  };

  /**
  * Check if an image has loaded without errors.
  *
  * @param {HTMLImageElement} imgElement an image.
  * @returns {boolean} true if this image has loaded without errors.
  */
  DragDropOntoImageQuestion.prototype.imageIsLoaded = function(
     imgElement) {
    return imgElement.complete && imgElement.naturalHeight !== 0;
```

```
};

/**
* Set up the question, once all images have been loaded.
*/
DragDropOntoImageQuestion.prototype.setupQuestion = function() {
  this.resizeAllDragsAndDrops();
  this.cloneDrags();
  this.positionDragsAndDrops();
 M.util.js_complete('qtype_ddlu-init-' + this.containerId);
};

/**
* In each group, resize all the items to be the same size.
*/
DragDropOntoImageQuestion.prototype.resizeAllDragsAndDrops =
    function() {
  var thisQ = this;
  this.getRoot().find('.draghomes > div').each(function(i, node) {
    thisQ.resizeAllDragsAndDropsInGroup(
    thisQ.getClassnameNumericSuffix($(node), 'dragitemgroup'));
  });
};

/**
* In a given group, set all the drags and drops to be the same
    size.
*
* @param {int} group the group number.
*/
DragDropOntoImageQuestion.prototype.resizeAllDragsAndDropsInGroup
   = function(group) {
  var root = this.getRoot(),
  dragHomes = root.find('.dragitemgroup' + group + ' .draghome'),
  maxWidth = 0,
  maxHeight = 0;

  // Find the maximum size of any drag in this groups.
  dragHomes.each(function(i, drag) {
    maxWidth = Math.max(maxWidth, Math.ceil(drag.offsetWidth));
    maxHeight = Math.max(maxHeight, Math.ceil(drag.offsetHeight));
  });

  // The size we will want to set is a bit bigger than this.
  maxWidth += 10;
  maxHeight += 10;
```

```
        // Set each drag home to that size.
        dragHomes.each(function(i, drag) {
          var left = Math.round((maxWidth - drag.offsetWidth) / 2),
          top = Math.floor((maxHeight - drag.offsetHeight) / 2);
          // Set top and left padding so the item is centred.
          $(drag).css({
            'padding-left': left + 'px',
            'padding-right': (maxWidth - drag.offsetWidth - left) + 'px
                ',
            'padding-top': top + 'px',
            'padding-bottom': (maxHeight - drag.offsetHeight - top) + '
              px'
          });
        });

        // Create the drops and make them the right size.
        for (var i in this.places) {
          if (!this.places.hasOwnProperty((i))) {
            continue;
          }
          var place = this.places[i],
          label = place.text;
          if (parseInt(place.group) !== group) {
            continue;
          }
          if (label === '') {
            label = M.util.get_string('blank', 'qtype_ddlu');
          }
          root.find('.dropzones').append('<div class="dropzone active
              group' + place.group +
          ' place' + i + '" tabindex="0">' +
          '<span class="accesshide">' + label + '</span> </div>');
          root.find('.dropzone.place' + i).width(maxWidth - 2).height(
              maxHeight - 2);
        }
      };

      /**
      * Invisible 'drag homes' are output by the renderer. These have
          the same properties
      * as the drag items but are invisible. We clone these invisible
          elements to make the
      * actual drag items.
      */
      DragDropOntoImageQuestion.prototype.cloneDrags = function() {
```

```
    var thisQ = this;
    thisQ.getRoot().find('.draghome').each(function(index, dragHome)
        {
      var drag = $(dragHome);
      var placeHolder = drag.clone();
      placeHolder.removeClass();
      placeHolder.addClass('draghome choice' +
      thisQ.getChoice(drag) + ' group' +
      thisQ.getGroup(drag) + ' dragplaceholder');
      drag.before(placeHolder);
    });
  };


  /**
  * Update the position of drags.
  */
  DragDropOntoImageQuestion.prototype.positionDragsAndDrops =
     function() {
    var thisQ = this,
    root = this.getRoot(),
    bgRatio = this.bgRatio();

    // Move the drops into position.
    root.find('.ddarea .dropzone').each(function(i, dropNode) {
      var drop = $(dropNode),
      place = thisQ.places[thisQ.getPlace(drop)];
      // The xy values come from PHP as strings, so we need parseInt
          to stop JS doing string concatenation.
      drop.css('left', parseInt(place.xy[0]) * bgRatio)
      .css('top', parseInt(place.xy[1]) * bgRatio);
      drop.data('originX', parseInt(place.xy[0]))
      .data('originY', parseInt(place.xy[1]));
      thisQ.handleElementScale(drop, 'left top');
    });

    // First move all items back home.
    root.find('.draghome').not('.dragplaceholder').each(function(i,
       dragNode) {
      var drag = $(dragNode),
      currentPlace = thisQ.getClassnameNumericSuffix(drag, 'inplace
          ');
      drag.addClass('unplaced')
      .removeClass('placed');
      drag.removeAttr('tabindex');
      if (currentPlace !== null) {
```

```
        drag.removeClass('inplace' + currentPlace);
      }
    });

    // Then place the ones that should be placed.
    root.find('input.placeinput').each(function(i, inputNode) {
      var input = $(inputNode),
      choice = input.val();
      if (choice.length === 0 || (choice.length > 0 && choice ===
          '0')) {
        // No item in this place.
        return;
      }

      var place = thisQ.getPlace(input);
      // Get the unplaced drag.
      var unplacedDrag = thisQ.getUnplacedChoice(thisQ.getGroup(
          input), choice);
      // Get the clone of the drag.
      var hiddenDrag = thisQ.getDragClone(unplacedDrag);
      if (hiddenDrag.length) {
        hiddenDrag.addClass('active');

      }

      // Send the drag to drop.
      var drop = root.find('.dropzone.place' + place);
      thisQ.sendDragToDrop(unplacedDrag, drop);
      var page = document.location.href;
      if(!page.includes('review')){
        var an = unplacedDrag.attr('class');
        document.getElementsByClassName(an)[0].setAttribute('hidden
            ',true);
      }
    });
  };

  /**
  * Handles the start of dragging an item.
  *
  * @param {Event} e the touch start or mouse down event.
  */
  DragDropOntoImageQuestion.prototype.handleDragStart = function(e)
      {
    var thisQ = this,
    drag = $(e.target).closest('.draghome'),
```

```
260       currentIndex = this.calculateZIndex(),
          newIndex = currentIndex + 2;

          var info = dragDrop.prepare(e);
          if (!info.start || drag.hasClass('beingdragged')) {
265         return;
          }

          drag.addClass('beingdragged').css('transform', '').css('z-index
              ', newIndex);
          var currentPlace = this.getClassnameNumericSuffix(drag, 'inplace
              ');
270       if (currentPlace !== null) {
            this.setInputValue(currentPlace, 0);
            drag.removeClass('inplace' + currentPlace);
            var hiddenDrop = thisQ.getDrop(drag, currentPlace);
            if (hiddenDrop.length) {
275           hiddenDrop.addClass('active');
              drag.offset(hiddenDrop.offset());
            }
          } else {
            var hiddenDrag = thisQ.getDragClone(drag);
280         if (hiddenDrag.length) {
              hiddenDrag.addClass('active');
              drag.offset(hiddenDrag.offset());
            }
          }
285
          dragDrop.start(e, drag, function(x, y, drag) {
            thisQ.dragMove(x, y, drag);
          }, function(x, y, drag) {
            thisQ.dragEnd(x, y, drag);
290       });
        };

        /**
        * Called whenever the currently dragged items moves.
295     *
        * @param {Number} pageX the x position.
        * @param {Number} pageY the y position.
        * @param {jQuery} drag the item being moved.
        */
300     DragDropOntoImageQuestion.prototype.dragMove = function(pageX,
            pageY, drag) {
          var thisQ = this,
          highlighted = false;
```

```
        this.getRoot().find('.dropzone.group' + this.getGroup(drag)).
            each(function(i, dropNode) {
          var drop = $(dropNode);
          if (thisQ.isPointInDrop(pageX, pageY, drop) && !highlighted) {
            highlighted = true;
            drop.addClass('valid-drag-over-drop');
          } else {
            drop.removeClass('valid-drag-over-drop');
          }
        });
        this.getRoot().find('.draghome.placed.group' + this.getGroup(
            drag)).not('.beingdragged').each(function(i, dropNode) {
          var drop = $(dropNode);
          if (thisQ.isPointInDrop(pageX, pageY, drop) && !highlighted &&
              !thisQ.isDragSameAsDrop(drag, drop)) {
            highlighted = true;
            drop.addClass('valid-drag-over-drop');
          } else {
            drop.removeClass('valid-drag-over-drop');
          }
        });
      };

      /**
      * Called when user drops a drag item.
      *
      * @param {Number} pageX the x position.
      * @param {Number} pageY the y position.
      * @param {jQuery} drag the item being moved.
      */
      DragDropOntoImageQuestion.prototype.dragEnd = function(pageX,
          pageY, drag) {
        var thisQ = this,
        root = this.getRoot(),
        placed = false;

        // Looking for drag that was dropped on a dropzone.
        root.find('.dropzone.group' + this.getGroup(drag)).each(function
            (i, dropNode) {
          var drop = $(dropNode);
          if (!thisQ.isPointInDrop(pageX, pageY, drop)) {
            // Not this drop.
            return true;
          }

          // Now put this drag into the drop.
```

```
        drop.removeClass('valid-drag-over-drop');
        thisQ.sendDragToDrop(drag, drop);
        placed = true;
        //СОКРЫТИЕ КАРТОЧКИ И ОБНОВЛЕНИЕ КОЛИЧЕСТВА
        var an = drag.attr('class');
        document.getElementsByClassName(an)[0].setAttribute('hidden',
            true);
        var num = document.getElementById("countDrop").innerHTML;
        document.getElementById("countDrop").innerHTML =  parseInt(num
            ) + parseInt(1);
        return false; // Stop the each() here.
      });

      if (!placed) {
        // Looking for that was dropped on a placed drag.
        root.find('.draghome.placed.group' + this.getGroup(drag)).not
            ('.beingdragged').each(function(i, placedNode) {
          var placedDrag = $(placedNode);
          if (!thisQ.isPointInDrop(pageX, pageY, placedDrag) || thisQ.
              isDragSameAsDrop(drag, placedDrag)) {
            // Not this placed drag.
            return true;
          }

          // Now put this drag into the drop.
          placedDrag.removeClass('valid-drag-over-drop');
          var currentPlace = thisQ.getClassnameNumericSuffix(
              placedDrag, 'inplace');
          var drop = thisQ.getDrop(drag, currentPlace);
          thisQ.sendDragToDrop(drag, drop);
          placed = true;
          return false; // Stop the each() here.
        });
      }

      if (!placed) {
        this.sendDragHome(drag);
      }
    };

    /**
    * Animate a drag item into a given place (or back home).
    *
    * @param {jQuery|null} drag the item to place. If null, clear the
        place.
    * @param {jQuery} drop the place to put it.
```

```
 */
DragDropOntoImageQuestion.prototype.sendDragToDrop = function(drag
    , drop) {
  // Is there already a drag in this drop? if so, evict it.
  var oldDrag = this.getCurrentDragInPlace(this.getPlace(drop));
  if (oldDrag.length !== 0) {
    oldDrag.addClass('beingdragged');
    oldDrag.offset(oldDrag.offset());
    var currentPlace = this.getClassnameNumericSuffix(oldDrag, '
        inplace');
    var hiddenDrop = this.getDrop(oldDrag, currentPlace);
    hiddenDrop.addClass('active');
    this.sendDragHome(oldDrag);
  }

  if (drag.length === 0) {
    this.setInputValue(this.getPlace(drop), 0);
    if (drop.data('isfocus')) {
      drop.focus();
    }
  } else {
    this.setInputValue(this.getPlace(drop), this.getChoice(drag));
    drag.removeClass('unplaced')
    .addClass('placed inplace' + this.getPlace(drop));

    drag.attr('tabindex', 0);
    this.animateTo(drag, drop);

  }
};

/**
* Animate a drag back to its home.
*
* @param {jQuery} drag the item being moved.
*/
DragDropOntoImageQuestion.prototype.sendDragHome = function(drag)
    {
  var currentPlace = this.getClassnameNumericSuffix(drag, 'inplace
      ');
  if (currentPlace !== null) {
    drag.removeClass('inplace' + currentPlace);
  }
  drag.data('unplaced', true);
```

```
425       this.animateTo(drag, this.getDragHome(this.getGroup(drag), this.
              getChoice(drag)));
          };

          /**
          * Handles keyboard events on drops.
430       *
          * Drops are focusable. Once focused, right/down/space switches to
              the next choice, and
          * left/up switches to the previous. Escape clear.
          *
          * @param {KeyboardEvent} e
435       */
          DragDropOntoImageQuestion.prototype.handleKeyPress = function(e) {
            var drop = $(e.target).closest('.dropzone');
            if (drop.length === 0) {
              var placedDrag = $(e.target);
440           var currentPlace = this.getClassnameNumericSuffix(placedDrag,
                  'inplace');
              if (currentPlace !== null) {
                drop = this.getDrop(placedDrag, currentPlace);
              }
            }
445         var currentDrag = this.getCurrentDragInPlace(this.getPlace(drop)
                ),
            nextDrag = $();

            switch (e.keyCode) {
              case keys.space:
450           case keys.arrowRight:
              case keys.arrowDown:
              nextDrag = this.getNextDrag(this.getGroup(drop), currentDrag);
              break;

455           case keys.arrowLeft:
              case keys.arrowUp:
              nextDrag = this.getPreviousDrag(this.getGroup(drop),
                  currentDrag);
              break;

460           case keys.escape:
              questionManager.isKeyboardNavigation = false;
              break;

              default:
465           questionManager.isKeyboardNavigation = false;
```

```
        return; // To avoid the preventDefault below.
      }

      if (nextDrag.length) {
        nextDrag.data('isfocus', true);
        nextDrag.addClass('beingdragged');
        var hiddenDrag = this.getDragClone(nextDrag);
        if (hiddenDrag.length) {
          hiddenDrag.addClass('active');
          nextDrag.offset(hiddenDrag.offset());
        }
      } else {
        drop.data('isfocus', true);
      }

      e.preventDefault();
      this.sendDragToDrop(nextDrag, drop);
    };

    /**
    * Choose the next drag in a group.
    *
    * @param {int} group which group.
    * @param {jQuery} drag current choice (empty jQuery if there isn't
        one).
    * @return {jQuery} the next drag in that group, or null if there
       wasn't one.
    */
    DragDropOntoImageQuestion.prototype.getNextDrag = function(group,
       drag) {
      var choice,
      numChoices = this.noOfChoicesInGroup(group);

      if (drag.length === 0) {
        choice = 1; // Was empty, so we want to select the first
           choice.
      } else {
        choice = this.getChoice(drag) + 1;
      }

      var next = this.getUnplacedChoice(group, choice);
      while (next.length === 0 && choice < numChoices) {
        choice++;
        next = this.getUnplacedChoice(group, choice);
      }
```

```
    return next;
};

/**
* Choose the previous drag in a group.
*
* @param {int} group which group.
* @param {jQuery} drag current choice (empty jQuery if there isn't
    one).
* @return {jQuery} the next drag in that group, or null if there
   wasn't one.
*/
DragDropOntoImageQuestion.prototype.getPreviousDrag = function(
   group, drag) {
    var choice;

    if (drag.length === 0) {
        choice = this.noOfChoicesInGroup(group);
    } else {
        choice = this.getChoice(drag) - 1;
    }

    var previous = this.getUnplacedChoice(group, choice);
    while (previous.length === 0 && choice > 1) {
        choice--;
        previous = this.getUnplacedChoice(group, choice);
    }

    // Does this choice exist?
    return previous;
};

/**
* Animate an object to the given destination.
*
* @param {jQuery} drag the element to be animated.
* @param {jQuery} target element marking the place to move it to.
*/
DragDropOntoImageQuestion.prototype.animateTo = function(drag,
   target) {
    var currentPos = drag.offset(),
    targetPos = target.offset(),
    thisQ = this;

    M.util.js_pending('qtype_ddlu-animate-' + thisQ.containerId);
```

```
        // Animate works in terms of CSS position, whereas locating an
           object
        // on the page works best with jQuery offset() function. So, to
           get
        // the right target position, we work out the required change in
        // offset() and then add that to the current CSS position.
        drag.animate(
        {
          left: parseInt(drag.css('left')) + targetPos.left - currentPos
             .left,
          top: parseInt(drag.css('top')) + targetPos.top - currentPos.
             top
        },
        {
          duration: 'fast',
          done: function() {
            $('body').trigger('qtype_ddlu-dragmoved', [drag, target,
               thisQ]);
            M.util.js_complete('qtype_ddlu-animate-' + thisQ.containerId
               );
          }
        }
        );
      };

      /**
      * Detect if a point is inside a given DOM node.
      *
      * @param {Number} pageX the x position.
      * @param {Number} pageY the y position.
      * @param {jQuery} drop the node to check (typically a drop).
      * @return {boolean} whether the point is inside the node.
      */
      DragDropOntoImageQuestion.prototype.isPointInDrop = function(pageX
         , pageY, drop) {
        var position = drop.offset();
        if (drop.hasClass('draghome')) {
          return pageX >= position.left && pageX < position.left + drop.
             outerWidth()
           && pageY >= position.top && pageY < position.top + drop.
             outerHeight();
        }
        return pageX >= position.left && pageX < position.left + drop.
           width()
         && pageY >= position.top && pageY < position.top + drop.height()
           ;
```

```
    };

    /**
    * Set the value of the hidden input for a place, to record what is
        currently there.
    *
    * @param {int} place which place to set the input value for.
    * @param {int} choice the value to set.
    */
    DragDropOntoImageQuestion.prototype.setInputValue = function(place
        , choice) {
      this.getRoot().find('input.placeinput.place' + place).val(choice
          );
    };

    /**
    * Get the outer div for this question.
    *
    * @returns {jQuery} containing that div.
    */
    DragDropOntoImageQuestion.prototype.getRoot = function() {
      return $(document.getElementById(this.containerId));
    };

    /**
    * Get the img that is the background image.
    * @returns {jQuery} containing that img.
    */
    DragDropOntoImageQuestion.prototype.bgImage = function() {
      return this.getRoot().find('img.dropbackground');
    };

    /**
    * Get drag home for a given choice.
    *
    * @param {int} group the group.
    * @param {int} choice the choice number.
    * @returns {jQuery} containing that div.
    */
    DragDropOntoImageQuestion.prototype.getDragHome = function(group,
        choice) {
      if (!this.getRoot().find('.draghome.dragplaceholder.group' +
          group + '.choice' + choice).is(':visible')) {
        return this.getRoot().find('.dragitemgroup' + group +
        // ' .draghome.infinite' +
        '.choice' + choice +
```

```
625         '.group' + group);
        }
        return this.getRoot().find('.draghome.dragplaceholder.group' +
            group + '.choice' + choice);
    };

630 /**
    * Get an unplaced choice for a particular group.
    *
    * @param {int} group the group.
    * @param {int} choice the choice number.
635 * @returns {jQuery} jQuery wrapping the unplaced choice. If there
     isn't one, the jQuery will be empty.
    */
    DragDropOntoImageQuestion.prototype.getUnplacedChoice = function(
        group, choice) {
        return this.getRoot().find('.ddarea .draghome.group' + group +
            '.choice' + choice + '.unplaced').slice(0, 1);
    };
640
    /**
    * Get the drag that is currently in a given place.
    *
    * @param {int} place the place number.
645 * @return {jQuery} the current drag (or an empty jQuery if none).
    */
    DragDropOntoImageQuestion.prototype.getCurrentDragInPlace =
        function(place) {
        return this.getRoot().find('.ddarea .draghome.inplace' + place);
    };
650
    /**
    * Return the number of blanks in a given group.
    *
    * @param {int} group the group number.
655 * @returns {int} the number of drops.
    */
    DragDropOntoImageQuestion.prototype.noOfDropsInGroup = function(
        group) {
        return this.getRoot().find('.dropzone.group' + group).length;
    };
660
    /**
    * Return the number of choices in a given group.
    *
    * @param {int} group the group number.
```

```
665     * @returns {int} the number of choices.
        */
        DragDropOntoImageQuestion.prototype.noOfChoicesInGroup = function(
            group) {
          return this.getRoot().find('.dragitemgroup' + group + ' .
              draghome').length;
        };
670
        /**
        * Return the number at the end of the CSS class name with the
            given prefix.
        *
        * @param {jQuery} node
675     * @param {String} prefix name prefix
        * @returns {Number|null} the suffix if found, else null.
        */
        DragDropOntoImageQuestion.prototype.getClassnameNumericSuffix =
            function(node, prefix) {
          var classes = node.attr('class');
680       if (classes !== '') {
            var classesArr = classes.split(' ');
            for (var index = 0; index < classesArr.length; index++) {
              var patt1 = new RegExp('^' + prefix + '([0-9])+$');
              if (patt1.test(classesArr[index])) {
685             var patt2 = new RegExp('([0-9])+$');
                var match = patt2.exec(classesArr[index]);
                return Number(match[0]);
              }
            }
690       }
          return null;
        };

        /**
695     * Get the choice number of a drag.
        *
        * @param {jQuery} drag the drag.
        * @returns {Number} the choice number.
        */
700     DragDropOntoImageQuestion.prototype.getChoice = function(drag) {
          return this.getClassnameNumericSuffix(drag, 'choice');
        };

        /**
705     * Given a DOM node that is significant to this question
        * (drag, drop, ...) get the group it belongs to.
```

```
     *
     * @param {jQuery} node a DOM node.
     * @returns {Number} the group it belongs to.
710  */
     DragDropOntoImageQuestion.prototype.getGroup = function(node) {
       return this.getClassnameNumericSuffix(node, 'group');
     };

715  /**
     * Get the place number of a drop, or its corresponding hidden
        input.
     *
     * @param {jQuery} node the DOM node.
     * @returns {Number} the place number.
720  */
     DragDropOntoImageQuestion.prototype.getPlace = function(node) {
       return this.getClassnameNumericSuffix(node, 'place');
     };

725  /**
     * Get drag clone for a given drag.
     *
     * @param {jQuery} drag the drag.
     * @returns {jQuery} the drag's clone.
730  */
     DragDropOntoImageQuestion.prototype.getDragClone = function(drag)
        {
       return this.getRoot().find('.dragitemgroup' +
       this.getGroup(drag) +
       ' .draghome' +
735    '.choice' + this.getChoice(drag) +
       '.group' + this.getGroup(drag) +
       '.dragplaceholder');
     };

740  /**
     * Get drop for a given drag and place.
     *
     * @param {jQuery} drag the drag.
     * @param {Integer} currentPlace the current place of drag.
745  * @returns {jQuery} the drop's clone.
     */
     DragDropOntoImageQuestion.prototype.getDrop = function(drag,
        currentPlace) {
       return this.getRoot().find('.dropzone.group' + this.getGroup(
          drag) + '.place' + currentPlace);
```

```
    };

    /**
    * Handle when the window is resized.
    */
    DragDropOntoImageQuestion.prototype.handleResize = function() {
      var thisQ = this,
      bgRatio = this.bgRatio();
      if (this.isPrinting) {
        bgRatio = 1;
      }

      this.getRoot().find('.ddarea .dropzone').each(function(i,
          dropNode) {
        $(dropNode)
        .css('left', parseInt($(dropNode).data('originX')) *
            parseFloat(bgRatio))
        .css('top', parseInt($(dropNode).data('originY')) * parseFloat
            (bgRatio));
        thisQ.handleElementScale(dropNode, 'left top');
      });

      this.getRoot().find('div.droparea .draghome').not('.beingdragged
          ').each(function(key, drag) {
        $(drag)
        .css('left', parseFloat($(drag).data('originX')) * parseFloat(
            bgRatio))
        .css('top', parseFloat($(drag).data('originY')) * parseFloat(
            bgRatio));
        thisQ.handleElementScale(drag, 'left top');
      });
    };

    /**
    * Return the background ratio.
    *
    * @returns {number} Background ratio.
    */
    DragDropOntoImageQuestion.prototype.bgRatio = function() {
      var bgImg = this.bgImage();
      var bgImgNaturalWidth = bgImg.get(0).naturalWidth;
      var bgImgClientWidth = bgImg.width();

      return bgImgClientWidth / bgImgNaturalWidth;
    };
```

```
    /**
790 * Scale the drag if needed.
    *
    * @param {jQuery} element the item to place.
    * @param {String} type scaling type
    */
795 DragDropOntoImageQuestion.prototype.handleElementScale = function(
       element, type) {
      var bgRatio = parseFloat(this.bgRatio());
      if (this.isPrinting) {
        bgRatio = 1;
      }
800   $(element).css({
        '-webkit-transform': 'scale(' + bgRatio + ')',
        '-moz-transform': 'scale(' + bgRatio + ')',
        '-ms-transform': 'scale(' + bgRatio + ')',
        '-o-transform': 'scale(' + bgRatio + ')',
805     'transform': 'scale(' + bgRatio + ')',
        'transform-origin': type
      });
    };

810 /**
    * Calculate z-index value.
    *
    * @returns {number} z-index value
    */
815 DragDropOntoImageQuestion.prototype.calculateZIndex = function() {
      var zIndex = 0;
      this.getRoot().find('.ddarea .dropzone, div.droparea .draghome')
         .each(function(i, dropNode) {
        dropNode = $(dropNode);
        // Note that webkit browsers won't return the z-index value
           from the CSS stylesheet
820     // if the element doesn't have a position specified. Instead
           it'll return "auto".
        var itemZIndex = dropNode.css('z-index') ? parseInt(dropNode.
           css('z-index')) : 0;

        if (itemZIndex > zIndex) {
          zIndex = itemZIndex;
825     }
      });

      return zIndex;
    };
```

```
830
    /**
    * Check that the drag is drop to it's clone.
    *
    * @param {jQuery} drag The drag.
835 * @param {jQuery} drop The drop.
    * @returns {boolean}
    */
    DragDropOntoImageQuestion.prototype.isDragSameAsDrop = function(
       drag, drop) {
      return this.getChoice(drag) === this.getChoice(drop) && this.
         getGroup(drag) === this.getGroup(drop);
840 };

    /**
    * Singleton object that handles all the DragDropOntoImageQuestions
    * on the page, and deals with event dispatching.
845 * @type {Object}
    */
    var questionManager = {

      /**
850   * {boolean} ensures that the event handlers are only initialised
          once per page.
      */
      eventHandlersInitialised: false,

      /**
855   * {Object} ensures that the drag event handlers are only
         initialised once per question,
      * indexed by containerId (id on the .que div).
      */
      dragEventHandlersInitialised: {},

860   /**
      * {boolean} is printing or not.
      */
      isPrinting: false,

865   /**
      * {boolean} is keyboard navigation or not.
      */
      isKeyboardNavigation: false,

870   /**
```

```
     * {Object} all the questions on this page, indexed by
        containerId (id on the .que div).
     */
     questions: {}, // An object containing all the information about
         each question on the page.

875  /**
     * Initialise one question.
     *
     * @method
     * @param {String} containerId the id of the div.que that
        contains this question.
880  * @param {boolean} readOnly whether the question is read-only.
     * @param {Array} places data.
     */
     init: function(containerId, readOnly, places) {
       questionManager.questions[containerId] =
885    new DragDropOntoImageQuestion(containerId, readOnly, places);
       if (!questionManager.eventHandlersInitialised) {
         questionManager.setupEventHandlers();
         questionManager.eventHandlersInitialised = true;
       }
890    if (!questionManager.dragEventHandlersInitialised.
          hasOwnProperty(containerId)) {
         questionManager.dragEventHandlersInitialised[containerId] =
            true;
         // We do not use the body event here to prevent the other
            event on Mobile device, such as scroll event.
         var questionContainer = document.getElementById(containerId)
            ;
         if (questionContainer.classList.contains('ddlu') &&
895      !questionContainer.classList.contains('qtype_ddlu-readonly')
            ) {
           // TODO: Convert all the jQuery selectors and events to
              native Javascript.
           questionManager.addEventHandlersToDrag($(questionContainer
              ).find('.draghome'));
         }
       }
900  },

     /**
     * Set up the event handlers that make this question type work. (
        Done once per page.)
     */
905  setupEventHandlers: function() {
```

```
        $('body')
        .on('keydown',
        '.que.ddlu:not(.qtype_ddlu-readonly) .dropzones .dropzone',
        questionManager.handleKeyPress)
        .on('keydown',
        '.que.ddlu:not(.qtype_ddlu-readonly) .draghome.placed:not(.
            beingdragged)',
        questionManager.handleKeyPress)
        .on('qtype_ddlu-dragmoved', questionManager.handleDragMoved);
        $(window).on('resize', function() {
          questionManager.handleWindowResize(false);
        });
        window.addEventListener('beforeprint', function() {
          questionManager.isPrinting = true;
          questionManager.handleWindowResize(questionManager.
              isPrinting);
        });
        window.addEventListener('afterprint', function() {
          questionManager.isPrinting = false;
          questionManager.handleWindowResize(questionManager.
              isPrinting);
        });
        setTimeout(function() {
          questionManager.fixLayoutIfThingsMoved();
        }, 100);
      },

      /**
      * Binding the drag/touch event again for newly created element.
      *
      * @param {jQuery} element Element to bind the event
      */
      addEventHandlersToDrag: function(element) {
        // Unbind all the mousedown and touchstart events to prevent
            double binding.
        element.unbind('mousedown touchstart');
        element.on('mousedown touchstart', questionManager.
            handleDragStart);
      },

      /**
      * Handle mouse down / touch start events on drags.
      * @param {Event} e the DOM event.
      */
      handleDragStart: function(e) {
        e.preventDefault();
```

```
    var question = questionManager.getQuestionForEvent(e);
    if (question) {
      question.handleDragStart(e);
    }
  },

  /**
  * Handle key down / press events on drags.
  * @param {KeyboardEvent} e
  */
  handleKeyPress: function(e) {
    if (questionManager.isKeyboardNavigation) {
      return;
    }
    questionManager.isKeyboardNavigation = true;
    var question = questionManager.getQuestionForEvent(e);
    if (question) {
      question.handleKeyPress(e);
    }
  },

  /**
  * Handle when the window is resized.
  * @param {boolean} isPrinting
  */
  handleWindowResize: function(isPrinting) {
    for (var containerId in questionManager.questions) {
      if (questionManager.questions.hasOwnProperty(containerId)) {
        questionManager.questions[containerId].isPrinting =
            isPrinting;
        questionManager.questions[containerId].handleResize();
      }
    }
  },

  /**
  * Sometimes, despite our best efforts, things change in a way
     that cannot
  * be specifically caught (e.g. dock expanding or collapsing in
     Boost).
  * Therefore, we need to periodically check everything is in the
     right position.
  */
  fixLayoutIfThingsMoved: function() {
    this.handleWindowResize(questionManager.isPrinting);
```

```
        // We use setTimeout after finishing work, rather than
           setInterval,
        // in case positioning things is slow. We want 100 ms gap
        // between executions, not what setInterval does.
        setTimeout(function() {
          questionManager.fixLayoutIfThingsMoved(questionManager.
             isPrinting);
        }, 100);
      },

      /**
      * Handle when drag moved.
      *
      * @param {Event} e the event.
      * @param {jQuery} drag the drag
      * @param {jQuery} target the target
      * @param {DragDropOntoImageQuestion} thisQ the question.
      */
      handleDragMoved: function(e, drag, target, thisQ) {
        drag.removeClass('beingdragged').css('z-index', '');
        drag.css('top', target.position().top).css('left', target.
           position().left);
        target.after(drag);
        target.removeClass('active');
        if (typeof drag.data('unplaced') !== 'undefined' && drag.data
           ('unplaced') === true) {
          drag.removeClass('placed').addClass('unplaced');
          drag.removeAttr('tabindex');
          drag.removeData('unplaced');
          drag.css('top', '')
          .css('left', '')
          .css('transform', '');

        } else {
          drag.data('originX', target.data('originX')).data('originY',
              target.data('originY'));
          thisQ.handleElementScale(drag, 'left top');
        }
        if (typeof drag.data('isfocus') !== 'undefined' && drag.data('
           isfocus') === true) {
          drag.focus();
          drag.removeData('isfocus');
        }
        if (typeof target.data('isfocus') !== 'undefined' && target.
           data('isfocus') === true) {
          target.removeData('isfocus');
```

```
        }
        if (questionManager.isKeyboardNavigation) {
          questionManager.isKeyboardNavigation = false;
        }
      },

      /**
      * Given an event, work out which question it effects.
      * @param {Event} e the event.
      * @returns {DragDropOntoImageQuestion|undefined} The question,
       or undefined.
      */
      getQuestionForEvent: function(e) {
        var containerId = $(e.currentTarget).closest('.que.ddlu').attr
           ('id');
        return questionManager.questions[containerId];
      }
    };

    /**
    * @alias module:qtype_ddlu/question
    */
    return {
      init: questionManager.init
    };
  });
```