//The main program which calls the three main components.

**Input:** A series of filenames as command line arguments.

**Output:** An xml representation of the game which is written to the disk.

function GameGenerator.main

xmlFiles : Map;

xmlFiles[CHARACTERS] ← args[0];

xmlFiles[LESSONS] ← args[1];

xmlFiles[CHALLENGES] ← args[2];

xmlFiles[LOCALE] ← args[3];

xmlFiles[SUBJECT] ← args[4];

xmlFiles[THEME] ← args[5];

gameGenerator : GameGenerator;

layers : Layers;

layers ← gameGenerator.loadXmlComponents(xmlFiles);

game : Game;

game ← gameGenerator.buildGame(layers);

**call** gameGenerator.exportGame(game, args[6]);

**Input:** A map of string to string representing the association of layers to their respective filenames in the repository.

**Output:** Layers, which is an object containing all layers or components used to build the game.

function loadXmlComponents

layers : Layers;

jaxbContext : JAXBContext;

file : File;

unmarshaller : Unmarshaller;

**for** layer ϵ layers – {lesson, challenge}

jaxbContext ← JAXBContext.newInstance(layer.class)

unmarshaller ← jaxbContext.createUnmarshaller();

file ← File(xmlFiles.layer);

layers.layer ← unmarshaller.unmarshal(file);

jaxbContext ← JAXBContext.newInstance(lesson.class)

unmarshaller ← jaxbContext.createUnmarshaller();

lessons : Lesson[];

**for** lessonFile ϵ xmlFiles.lessons

lesson : Lesson;

file ← File(lessonFile);

lesson ← unmarshaller.unmarshal(file);

lessons ← lessons ∪ {lesson};

jaxbContext ← JAXBContext.newInstance(challenge.class)

unmarshaller ← jaxbContext.createUnmarshaller();

challenges : Challenge[];

**for** challengeFile ϵ xmlFiles.challenges

challenge : Challenge;

file ← File(challengeFile);

challenge ← unmarshaller.unmarshal(file);

challenges ← challenges ∪ {challenge};

learningActs : LearningAct[];

**for** (lesson ϵ lessons) && (challenge ϵ challenges)

learningAct: LearningAct;

lessonActs : LessonAct[];

lessonAct : LessonAct;

lessonAct.lessonScreens ← lesson;

lessonAct.challengeScreens ← challenge;

lessonActs ← lessonActs ∪ {lessonAct};

learningAct.lessonActs ← lessonActs;

learningActs ← learningActs ∪ {learningAct};

layers. learningActs ← learningActs;

**call** wireUpLayers;

**Input:** The layers object containing all entities with all dependencies set.

**Output:** A Game object containing the built and assembled game.

function buildGame

game : Game;

game ← layers.getStructure().createGame();

**Input:** A Game object with a complete game and a filename where the game should be exported.

**Output:** An xml file representing the game which is written to the disk.

function exportGame

jaxbContext : JAXBContext;

jaxbContext ← JAXBContext.newInstance(Game.class);

marshaller : Marshaller;

marshaller ← jaxbContext.createMarshaller();

marshaller[Marshaller.JAXB\_FORMATTED\_OUTPUT] ← true;

file : File;

file ← new File(exportFilename);

**call** marshaller.marshal(game, file);

**Input:** All inputs are dependencies.

**Output:** A Game object representing the created game.

function createGame

acts : Act[];

screens : ScreenNode[];

screens ← theme.getIntro();

acts ← acts ∪ createActFromScreens(screens);

**for**(int i = 0; i < locale.getLearningActs().size(); i++)

screens ← locale.getAct(i);

acts ← acts ∪ createActFromScreens(screens);

screens ← theme.getOutro();

acts ← acts ∪ createActFromScreens(screens);

game : Game;

game.acts ← acts;

**call** wireUpActs(acts);

**return** game;

**Input:** The learning act id, and the screen type.

**Output:** A list of ScreenNode which represents the screens.

function buildScreens

lessonScreens : ScreenNode[];

currentScreen : UUID;

nextScreen : UUID;

currentScreen ← UUID.randomUUID();

themeStory : ThemeStory;

themeStory ← theme.getThemeStories()[learningActId];

themeStoryScreen : BaseScreen[];

**if** (screenType == ScreenType.LESSON\_STORY\_INTRO)

themeStoryScreen ← themeStory.getIntro();

**else**

screenTransitions[TransitionType.END\_OF\_STORY] ← currentScreen;

themeStoryScreen ← themeStory.getOutro();

**for** screen ϵ themeStoryScreen

nextScreen ← UUID.randomUUID();

lessonScreens ← lessonScreens ∪ buildScreen(learningActId, screen, localeScreens[screenType], currentScreen, nextScreen);

currentScreen ← nextScreen;

**if** (screenType == ScreenType.LESSON\_STORY\_INTRO)

screenTransitions[TransitionType.BEGINNING\_OF\_LESSON] ← nextScreen;

**return** lessonScreens;