Virtual Teacher/Mentor

High Level Design

COP4331C: Fall 2012

Modification history:

Version	Date	Who	Comment
v1.0	10/09/12	Stephen Caskey	Initial Version Compilation

Team Name: Team #14

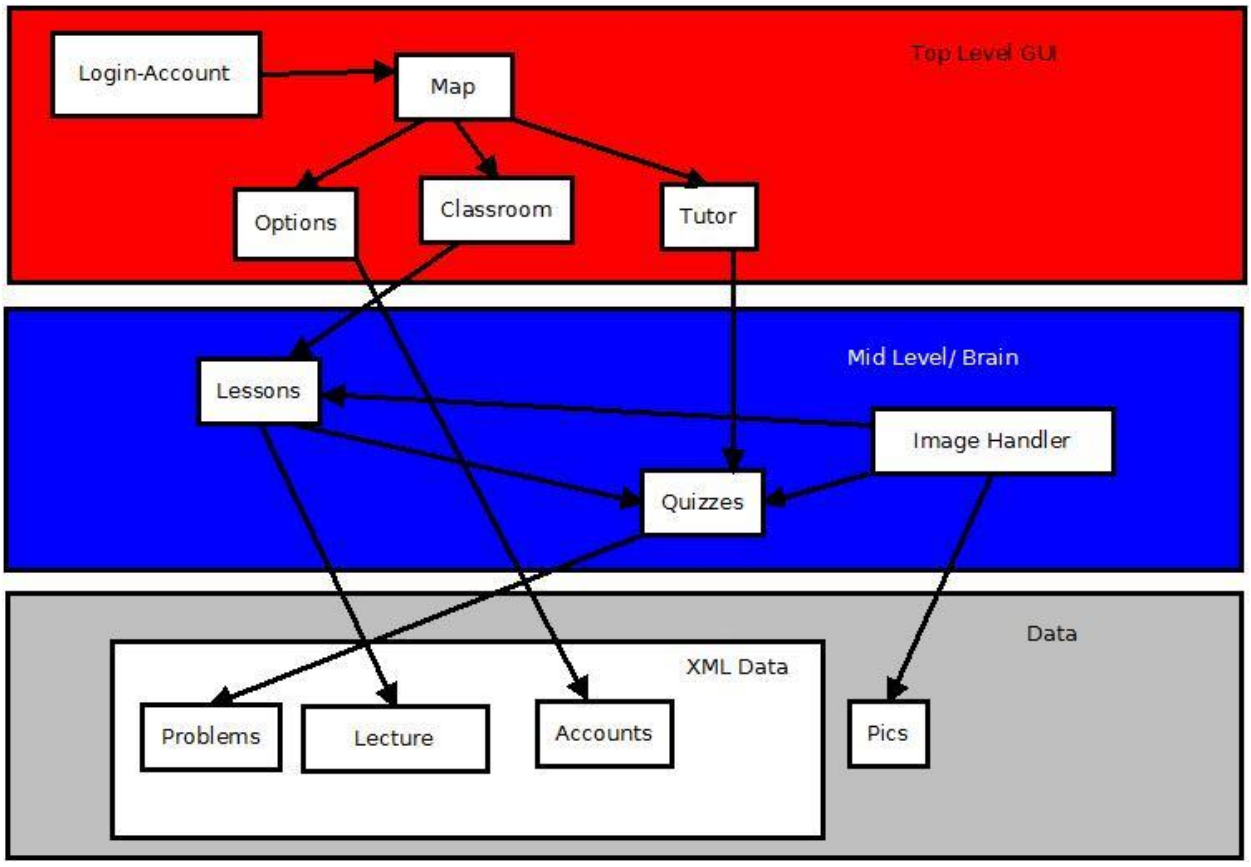
Team Members:

- Stephen Caskey - stevecask@knights.ucf.edu
- Anthony Delprete - adelprete@knights.ucf.edu
- Kevin Krutek - tronta@knights.ucf.edu
- Jason Shih - soulzityr@gmail.com
- Jason Shipman - thearmorywars@gmail.com
- Marlon Smith - marlonosmith@knights.ucf.edu

Contents of this Document

High Level Architecture:

Component Diagram:



Interface Detail:

The high level design is broken up into 3 layers Top layer, Middle layer, and Data layer.

Top/Gui Layer:

- **Login-Account:** The user will be greeted by this screen. Here they will choose to create an account or log in.
- **Map:** After logging in the user will go to the Map. Here they will see multiple options: Options, Classroom/School, tutor.
- **Options:** Here the user will be able to modify their account(name, age, etc)
- **Classroom:** When classroom is chosen they move down to the middle layer of our program to lectures. Whichever subject they choose the lecture of that day will be loaded.
- **Tutor:** With this option the user will receive some extra questions on a subject of their choosing. They count for nothing; Just practice.

Middle Layer:

- **Lessons:** Lessons is the real meat of the program. When called, it'll have to look up what School year it is, what day it is and what subject lecture the user called for. Lessons delves into the lecture part of our Data layer to retrieve the right lesson.
- **Quizzes:** Quizzes can be called from either the Tutor or Lessons. Quizzes will have to see what school year it is, what day it is, and what subject is being worked on to grab the correct set of problems from Problems in our Data layer.
- **ImageHandler:** We will have an image handler to handle any images we may need to load at any given time.

Data Layer:

- **Problems:** Using XML we will store all of our problems/questions. They will be organized by school year and day.
- **Lecture:** Using XML we will store all of our lectures. They will be organized by school year and day.
- **Accounts:** Using XML we will store any accounts that were created by users.
- **Pics:** Pics will hold any pictures that need to be loaded by our program. They will be organized accordingly.

Design Issues:

Performance:

- Our team is new to using Java slick which is a 2D game development software. We are a little concerned about how smooth every transition will be from event to event. We are implementing a drop down menu that allows a user to jump back to a specific week and day in the school year and these changes will happen in real time. We aren't exactly sure how well this'll work or how well the program will perform doing this action.

Portability:

- The user should be able to put the entire game onto a flash drive and play it wherever they please. The problem with this is that with each new PC the PC will need to be Windows based and be able to run a JAVA program.

Testability:

- An issue with testing is that we literally have to test each problem with a correct answer and a wrong answer to fully test our system of any bugs in question answering. This may take quite a bit of time.

Technical Difficulties:

- We really want to implement two different lecture formats: Video based lecture and Text and Image based lecture. Using Java Slick we are unsure if we can put videos into our program, which may force us to scrap video lectures all together and have to come up with a new way to teach concepts such as math.

Design trade-offs:

- **Client – Server** will not be used. This program won't be using any server of any kind.

- **Peer to Peer** will not be used. Users won't be communicating with any other users. This is strictly a single player learning experience.

Architectural Design Choice Rationale:

- There isn't a single design choice presented to us that fits our program perfectly so we are going to have to pull features from a couple of different designs. A publish –Subscribe type design may work. That designs ability to have a shared repository for components to share persistent data will and the fact that components express interest in an event by subscribing to it will work well for this program. Parts of Layering will also work well for us. Our program is designed in a way that we have layers that delve deeper into our program until it reaches the data and then moves back up the layer set.

Technical Risks:

- This is a program that relies heavily on triggered events. We have to make sure that following each event trigger that the appropriate data is called upon and that depending on where the user is and being taken to, this data is displayed to the screen appropriately. This is the foundation that our program is built on so it's a huge risk we are taking relying on whether or not we can get data quickly and effectively back to the user.

Virtual Teacher/Mentor

Detailed Design

COP4331C: Fall 2012

Modification history:

Version	Date	Who	Comment
v1.0	10/09/12	Stephen Caskey	Initial Version Compilation

Team Name: Team #14

Team Members:

- Stephen Caskey - stevecask@knights.ucf.edu
- Anthony Delprete - adelprete@knights.ucf.edu
- Kevin Krutek - tronta@knights.ucf.edu
- Jason Shih - soulzityr@gmail.com
- Jason Shipman - thearmorywars@gmail.com
- Marlon Smith - marlonosmith@knights.ucf.edu

Contents of this Document

Design Issues:

Performance:

- The biggest performance issue our program faces is effectively retrieving the appropriate course data for each class. Each day in the school year holds a different set of data that corresponds to each of the subjects. We need to make sure that when the user wants to enter the math class on day 8 that they get the right math lecture and quiz for that particular day and that they get it fast. The constant retrieval of data from our database can prove to be costly and it's a great design issue.

Testability:

- Testing is may be difficult for us to do because the program is designed to be played in an ordered way. We will have to test every lecture and every question to see if they display properly and that questions are given valid answers. For each question we will need to try the correct answer and the wrong answer to make sure that each quiz is assessed properly.
- This program is also geared towards children from grades 1 to 5. We don't know anyone within that age range so it may be difficult to judge how effective this program is.

Portability:

- We really want this program to be portable but it's all going to depend on the size of the final product. The idea is to be able to store this program on a 4gb flash drive and take it wherever you like but our program has the potential to have a TON of data (pics, sounds, possibly videos, etc.) and the more we work on this assignment the less likely this'll work out.

Design Decisions:

- We are going to have an interactive "home" screen. That displays a map of a town that includes the School, Home(Tutor), Options, and Quit. Each option will be one of the houses in the town.
    - Risk: We are not artists so making the town appeal to kids may be difficult.

- Give the student the chance to skip a grade by taking a grades Final Exam test.

- Allow the student to go back to any day they have already done if they want to re-watch a lecture. It'll be good for the student to have a "refresher" if they feel they need it.

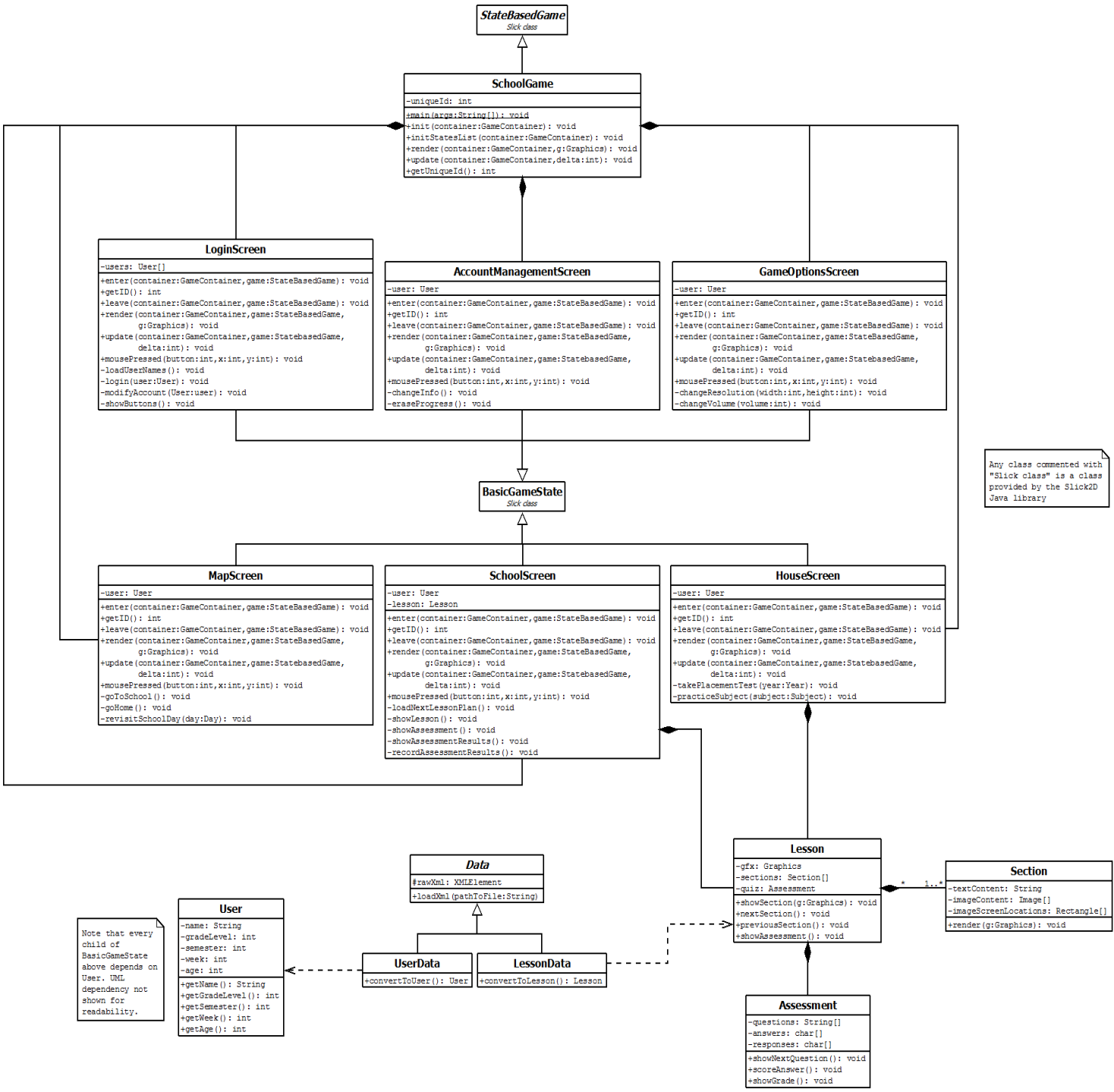
- Have the user pass the daily quiz to move on to the next day. If they fail they have to retake it until they pass.
    - Risk: The student might see repeat questions and know which answers are correct and which are wrong.

- The student will be able to be tutored in any subjects they need help in. This will generate new questions for the student to do that weren't in school.
    - Risk: Not having a large variety of questions. We want to avoid repeats.

- The program will have a child-like aesthetic. We need to appeal to the children and make it look fun.

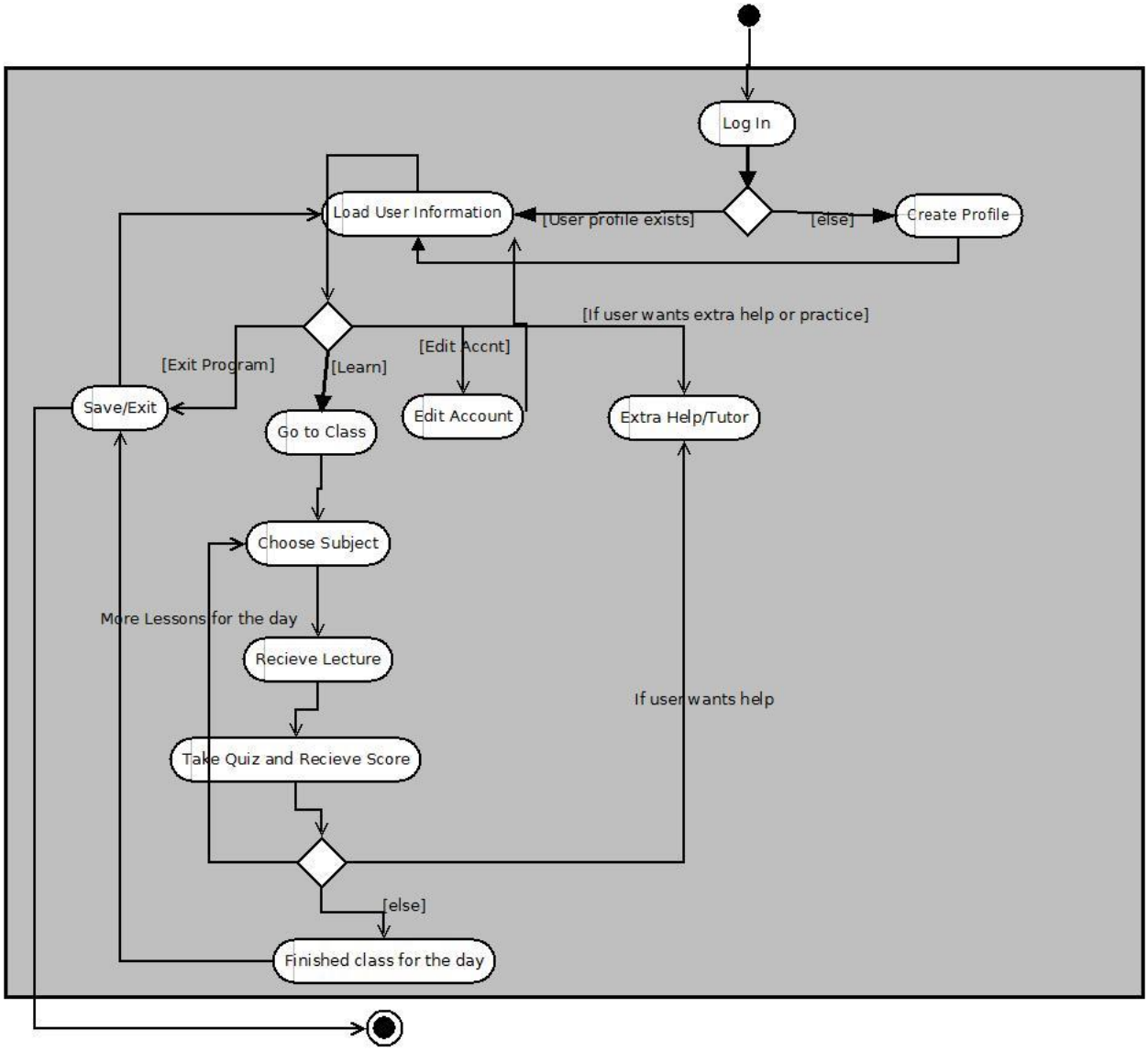
Detailed Design Information:

Class Diagram:



If this cannot be read, zoom in or please visit:
thearmorywars.github.com/VirtualTeacherGame/res/class_diagram.png

Activity Diagram:



Trace of Requirements to Design:

Functional Requirements:

Requirement Number	Requirement Description	Classes Used
3	The program shall accurately grade inputs to determine their accuracy, display corresponding messages, and modify grade values accordingly.	Assessment
5	The program shall have familiar account management actions. A user must be able to create, save, and delete their account.	LoginScreen AccountManagementScreen GameOptionsScreen
6	The program must teach subjects in the form of school simulation every day for a set of weeks. Subjects include Math, Science, History, and Language Arts.	Lesson SchoolScreen
7	A typical “day” in the program must have the user attend their classes and be tested on that day's material via homework or in-class assignments. If the student does poorly, they can re-visit the material and re-take the assessment.	SchoolScreen HouseScreen
8	Immediately following an assessment of any form, the program must provide the user with their results.	Assessment
9	When the school “week” ends, the user must see how they did over that week. If they want, they should be able to re-visit the activities for a day they did poorly on to improve their scores.	SchoolScreen HouseScreen
10	The program must auto-save the user's progress. This process should be invisible to the user.	UserData User

Interface Requirements:

Requirement Number	Requirement Description	Classes Used
11	The program shall take as inputs answers to questions posed to users and grade their accuracy before returning this grade to the user.	Assessment User
14	Questions and answers will interchange at a rate of 1:1, and all data will be managed locally, so no timing or servers are necessary.	LessonData

Users and Human Factors Requirements:

Requirement Number	Requirement Description	Classes Used
20	All users should require no training outside of a basic tutorial. All user functionality must be simple and explicit to accommodate this. The assumed skill level for users in fairly low.	SchoolGame
22	The program should be adequately simple and should require only minimal protection against misuse. The only realistic misuse to be expected is students potentially modifying their grades or looking into answers	UserData LessonData

Documentation Requirements:

Requirement Number	Requirement Description	Classes Used
23	The program shall record grades and print them out, potentially in the form of certifications or report cards for the benefits of students.	SchoolGame HouseScreen UserData

Data Requirements:

Requirement Number	Requirement Description	Classes Used
25	Data calculations will be kept simple with accuracy only to a few decimal places where needed. Grading is likely the most intensive part of this and will require only 2 points of accuracy. Math problems for students will used basic calculator math and will work primarily in integers.	Assessments LessonData UserData
26	The program must retain grades for students indefinitely.	UserData

Security Requirements:

Requirement Number	Requirement Description	Classes Used
28	Access to system information should be controlled for multiple users	AccountManagement UserData
29	One account's user data should be isolated from others	UserData
30	The system will save data to files and should not need to back up otherwise. No other precautions should be necessary.	UserData