Mark Sherman, Fred Martin, Larry Baldwin, James DeFilippo

Introduction

The rubric assumes that the person doing the scoring has background in computer science. Some of the rows describe advanced ideas that would only be meaningful to someone with a priori knowledge in the field, or knowledge acquired as a result of successfully completing a CS Principles-style intro course.

Broadly, the rubric provides three orthogonal ways of scoring: (1) What computational ideas are represented in the project code itself, and (2) (2) their appropriateness and (3) their correctness of these ideas, given a broader instructional context that a teacher may have. For each computational feature, these broad characteristics are treated separately.

For each feature, up to four categories are provided for characterizing its computational properties. These categories range from “feature is not present” (or “feature is minimally present,” for features that simply must exist given App Inventor) to a description of what constitutes advanced usage of the feature.

For each feature, after its computational properties are scored, the rater is invited to judge the appropriateness and correctness of the design. These judgments can only be made with the use of broader knowledge about the student’s developmental level, the instructional context, and the particular assignment for which the app being scored was developed. Rater should have this context (e.g., by being the course instructor) in order to make these judgments.

For example, consider the Screen Interface feature. All apps must have a screen, so the minimally-present category is described as *Single screen with five or fewer visual components*. The adjacent category is described as *Single screen with more than five visual components.* Both of these categories describe essentially static interfaces. The third and fourth categories then describe two degrees of complexity of dynamic interfaces: *Single screen with more than five visual components, some of which programmatically change state based on user interaction with the app* and *Two or more screens; screens may be implemented as screen components, or by programmatically changing visibility of groups of visual components*.

After characterizing the properties of the implementation, the rater should judge its appropriateness and correctness. The Screen Interface feature may be characterized as (appropriateness) too simple, just right, and too complex and (correctness) broken (e.g.; buttons not connected to code; interface crashes in some cases), almost works (minor UI problems), and correct.

App Title:

Rater:

Course:

Name(s) of Programmer(s):
(If this was a group project list the members of the group.)

Brief description of what the app is intended to do:

Institution:

Instructor(s):

Mark Sherman, Fred Martin, Larry Baldwin, James DeFilippo

	1	2	3	4	
Screen Interface	Single screen with five or fewer visual components.	Single screen with more than five visual components.	Single screen with more than five visual components, some of which programmatically change state based on user interaction with the app.	Two or more screens; screens may be implemented as screen components, or by programmatically changing visibility of groups of visual components.	*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness of UI implementation:* [ ] broken (e.g.; buttons not connected to code; interface crashes in some cases) [ ] almost works (minor UI problems) [ ] correct
Naming: **Components** **Variables** **Procedures**	Few or no names were changed from their defaults.	Approximately half of names have been changed from their defaults.	All or nearly all of names have been changed from their defaults.	Nearly all names have been changed and the given names are descriptive of their contents.	*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Events	Fewer than three types of event handlers. (Multiple buttons, all with "buttonx.onClick", are of the same type, and simple.)	Three or more types of event handlers; event handlers don't trigger each other. Blocks in event handlers don't enable or create future events, directly or indirectly (e.g. by changing global state).	One pair of interacting event handlers. One event handler contains blocks that create or enable another event. Examples are any use of an action with a callback event, enabling or disabling clocks programmatically.	More than one pair of interacting event handlers. (e.g., two pairs, or group of three or more interacting event handlers)	*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Procedural Abstraction	There are no procedures.	Repeatedly used code is placed in procedures.	Code is organized into functional modules by procedures.	Procedures may call other procedures, or recurse.	Appropriateness: [ ] too simple [ ] just right [ ] too complex Correctness: [ ] broken [ ] almost works [ ] correct

Mark Sherman, Fred Martin, Larry Baldwin, James DeFilippo

Data Abstraction: Variables & Constants	No data abstraction. Values are hard-coded.	Variables provide names to data, whose values change as the app runs, or may not (equivalent to a named constant).			Appropriateness: [ ] too simple [ ] just right [ ] too complex Correctness: [ ] broken [ ] almost works [ ] correct
Data Abstraction: Data Organization	Data are limited to scalar variables.	Data are stored in list(s), and operations are abstract over list(s).	Makes use of lists as data structures, packing multiple pieces of data into lists which conceptually represent something.		Appropriateness: [ ] too simple [ ] just right [ ] too complex Correctness: [ ] broken [ ] almost works [ ] correct
Component Abstraction	App does not use component abstraction.	App modifies or reads properties of components out of a list using an “any component” block.			*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Algorithms	Behaviors are simple, composed of individual and unrelated events.	Long or complex tasks are executed			*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Loops	No loops.	Simple loop, using a constant-value control values.	Loop is governed by data that may change, dynamic.	Loop uses control values that connect data together, across multiple lists, or addressing multiple structures.	*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Conditionals	No conditionals.	Conditionals use comparison of a variable value to a constant value.	Conditionals use comparison of two variable values.		*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct

Lists	No lists.	One single-dimensional list.	More than one independent, single-dimensional lists.	A list of tuples, or (equivalently) multiple corresponding lists (a "multi-dimensional list").	*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Data Persistence	Data are only stored in variables or UI component properties, and do not persist when app is closed.	Data persist beyond a single session of the app. (look for: tinydb, tinywebdb)			*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Data Sharing	No data sharing.	Shared data are limited to scalar variables. (such as a high score)	Shared data are stored in lists or structures. (such as a high score with name)	App's users can extend the types of data that are shared.	*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Public Web Services	No web services.	Reads data directly from online data source.	Reads and writes online data source.		*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Accelerometer & Orientation Sensors	No sensors used.	Accelerometer Shake gesture used to trigger events.	App makes decisions based on sensor data (e.g. controls a sprite).		*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct
Location Awareness	No location used.	Accesses location and immediately passes it to built-in features (such as maps)	Accesses location and stores it for later retrieval and use.	Inspects location data numerically, processes this data as a feature.	*Appropriateness:* [ ] too simple [ ] just right [ ] too complex *Correctness:* [ ] broken [ ] almost works [ ] correct

Overall assessment of App:

If this app has notable qualities or contains a good example of some aspect of Mobile Computational thinking, please describe it briefly:

To what extent did the app fulfill its intended purpose?

How would you rate the aesthetics of the app? How much effort was applied to the visual experience?

Additional comments: