SSUI Mobile Lab
Project 3: An FSM-Based Game Engine

Due: Wednesday, October 22nd by 11:59pm

Goal

In this assignment you will be implementing parts of a rudimentary game engine that is implemented using a table-based finite state machine. The program assignment is one demonstration of the value of finite state machines for managing user interfaces.

Project Overview

At the highest level, this game engine reads in an XML file that defines a table-based finite state machine (FSM), and implements that FSM. The XML specifies two top-level items: **Characters** which are the only objects that appear on the screen (though they do not always appear), and **Buttons**, which are identified by a string. Each character contains an FSM. While details are provided below, the basic idea is that a character can be manipulated by delivering events to it as specified in its FSM (for example, it can react to a button press), which can carry out actions.

Overview of Provided Files (don't edit any of these)

SSUIMobileLabGameEngineActivity.java	The activity that displays the game engine.
GameEnginePreBase.java	This abstract class overrides View and provides top-level functionality for the game engine. It contains an array of all **Characters** that are in the game and the dispatching of events. It also contains the code that carries out animations. **GameEngineBase**, which you are responsible for implementing, inherits from this.
GameCharacterPreBase.java	This abstract class contains code to manage each Character in the game. You will implement **GameCharacterBase**, which inherits from this method. Among other things, each Character knows what state it is currently in, and stores the FSM that governs its behavior.
FSMState.java	This class represents a state in the FSM. Each state contains an array of **FSMTransition**s, which it can use to respond to **FSMEvent**s.
FSMEvent.java (direct/indirect superclass to: **AnimEndEvent, AnimMoveEvent, AnimStartEvent, ButtonPressedEvent, DragMoveEvent, DragEndEvent, MessageEvent, TouchMoveEvent, TouchPressEvent, TouchReleaseEvent**	This class is the class from which all events are inherited. You are responsible for delivering these events to the proper character.

and XYEvent)
Types defined in **FSMEventType.java**

FSMTransition.java	An FSMTransition specifies: • What event triggers this transition through the **match(FSMEvent evt)** method. Note this will match all events, including button presses and messages. • What the target state of the transition is • What **FSMAction**s should be carried out on this transition. Note that a transition may have 0 or more actions associated with it.
FSMAction.java **(**direct/indirect superclass to: **ChangeImageAction, DebugAction, DropDragFocusAction, FollowEventAction, GetDragFocusAction, MoveIncAction, MoveToAction, RunAnimAction, SendMessageAction, StringAction, XYAction)** Types defined in **FSMActionType.java**	An FSMAction holds the information for actions that should be carried out on a transition. See more information below.

What You Need To Implement

To complete this programming assignment, you need to implement two classes:

GameEngineBase.java
This class is responsible for managing drawing of the game board, the translation of low-level events to higher-level FSMEvents, and the delivery of those events. The methods it should implement are:

onDraw(Canvas canv)	This method is responsible for drawing all Characters.
buttonHit(int buttonNumber)	This method is called whenever a button has been clicked by the user. The parameter is the index number of the button.
charactersUnder(RectF area)	Find and return the list of characters whose bounds overlap the given rectangular area. The characters (if any) in the list should be in reverse drawing order. That is the characters drawn later should appear earlier in the list.
dispatchPositionally(XYEvent evt)	Dispatch the given event to one character under the given x,y position. When multiple characters are under the position we offer it

	to them in reverse drawing order. As soon as a character takes the event (returns true from its deliverEvent() method) we stop offering it to others so that only one character gets the event.
dispatchPositionally(RectF inArea, FSMEvent evt)	Dispatch the given event to one character whose bounds overlap the given rectangle. When multiple characters are overlapped we offer it to them in reverse drawing order. As soon as a character takes the event (returns true from its deliverEvent() method) we stop offering it to others so that only one character gets the event.
dispatchDirect(int toChar, FSMEvent evt)	Dispatch the given event directly to the given character.
dispatchToAll(FSMEvent evt)	Dispatch the given event to all characters in reverse drawing order. This dispatch does not stop after the first character accepts the event, but instead always continues through the list of all characters.
dispatchTryAll(FSMEvent evt)	Attempt to dispatch the given event to all characters in reverse drawing order stopping as soon as some character takes the event (returns true from its deliverEvent() method).
dispatchDragFocus(FSMEvent evt)	Dispatch the given event to the current drag focus object (if any). If there is no current drag focus or the current drag focus object rejects the event (returns false from its deliverEvent() method), this method returns false. All events which contain an x,y position (i.e., subclasses of XYEvent) will have their x,y position adjusted by (-_grabPointX, -grabPointY) prior to being delivered (or more correctly a copy of the event will be adjusted and delivered). In this way the position indicated in the event will reflect where the top-left corner of the dragged character should be placed, rather than where the cursor was (which will normally be inside the character; specifically at a distance of (_grabPointX, _grabPointY) from the top-left of the object).
onTouchEvent(MotionEvent evt)	This method is called whenever a touch event occurs inside of the game view area (which does not include the buttons). You should be implementing code to translate

	these clicks to meaningful events and deliver these events. Hints: (1) the event you dispatch will depend on whether the MotionEvent is an ACTION_DOWN, ACTION_MOVE, or ACTION_UP. (2) You should use the dispatch methods you implemented above (e.g., dispatchTryAll()) and instantiate the appropriate events implemented for you in the ssuimobile.gameengine.event package (e.g. TouchReleaseEvent).

GameCharacterBase.java

This class is responsible for managing the character's FSM, responding to events, and drawing the character:

draw(Canvas canv)	This method is responsible for the character's bitmap. A character may or may not have a bitmap. If there is no bitmap, the character will not have anything visible on the game board (but that does not mean it is inactive). If a bitmap is drawn, it should be drawn with the top left corner at the x,y coordinates of the Character. However, the intrinsic dimensions of the bitmap should be respected, and the Character width and height should not cause the image to be clipped, nor transformed in any other way.
deliverEvent(FSMEvent event)	This method should be called by your code to deliver an event to a particular Character. The method returns true if the event is consumed and false if it is not. The Character should only consume an event if it has a transition with an event that matches that event.
makeFSMTransition (FSMTransition transition)	This method should be called by your code whenever a Transition is being followed. This method should make sure that any Actions on the transition are carried out (which may also require redraws) and should maintain what state has been transitioned to.

This is a description of the behavior that should occur for each action:

Action	Description
CHANGE_IMAGE	Causes the character to change its image to the parameterized bitmap. If the parameter is null, no image should be displayed.
MOVE_TO	Moves the character to exact x,y coordinates (no bounds checking)
MOVE_INC	Moves the character by the increment specified by x and y, relative to its current position (no bounds checking)
FOLLOW_EVENT_POSITION	Move the character to the x, y coordinates of the corresponding event
GET_DRAG_FOCUS	Cause the character to be the focus of a drag event
DROP_DRAG_FOCUS	Cause the character to release focus of the drag event
SEND_MESSAGE	Send the specified message to the specified character
DEBUG_MESSAGE	Write a debug message to the debug log with the tag "ssui"
RUN_ANIM	Cause the character to begin the specified animation

Testing and the XML File

An XML file is bundled with the distributed program assignment in res/raw/basic_fsmspec.xml. Provided is a **FSMxmlTranslator.java** file which reads in the XML file and calls the proper constructors to generate the FSM, so you do not need to implement that. While a simple XML file is provided, you will need to test your implementation by changing the XML file that specifies the state machine.

Here is a description of the XML format:

Tag	Attributes	Child Of
<ssuigamefsmspec> root tag for the XML File	none	Root
<character> Describes a particular character in the game.	**name** – String, unique to character all characters in the fsm **initbm** (optional)– String corresponding to the file name of the bitmap in res/layout to represent this character (may be null) **initx** – integer representing the initial x position of this character	ssuigamefsmspec

	inity – integer representing the initial y position of this character **initw** – integer representing the initial width of this character **inith** – integer representing the initial height of this character	
<fsm> The finite state machine that describes the behavior of this character. A character must have exactly one fsm	none	character
<state> A state in the finite state machine. An fsm may have zero or more states	**name** – string that identifies the state which is unique to the FSM it is in	fsm
<transition> A transition from the state it is a child of. A state may have zero or more transitions.	**targetState** – string that matches the name of the state that this transition goes to	state
<eventMatch> describes the event that is supposed to match this transition. A transition has exactly one eventMatch.	**type** – string that describes the type of event this transition should match on. Must be one of: NO_EVENT, BUTTON_PRESSED, ANIM_START, ANIM_MOVE, ANIM_END, MESSAGE_ARRIVED, TOUCH_PRESS, TOUCH_MOVE, TOUCH_RELEASE, DRAG_MOVE, DRAG_END **message** (only for MESSAGE_ARRIVED) – string that is the message that this transition responds to, which must exactly match the text of the sent message **buttonName** (only for BUTTON_PRESSED) – string that matches the name of the button that this transition responds to	transition
<action>	**type** – string that describes the type of action that this transition should cause to	transition

describes the action that is to be taken if this transition is matched. A transition may have zero or more actions.	happen, must be one of: CHANGE_IMAGE, MOVE_TO, MOVE_INC, FOLLOW_EVENT_POSITION, RUN_ANIM, GET_DRAG_FOCUS, DROP_DRAG_FOCUS, SEND_MESSAGE, DEBUG_MESSAGE **bitmap** (for CHANGE_IMAGE only) – String corresponding to the file name of the bitmap in res/layout to represent this character (may be null) **movingCharacterName** (for RUN_ANIM only) – String that matches the name of the character that should animate **targetCharacterName** (for RUN_ANIM only) – String that matches the name of the character that should animate **endMessage** (for RUN_ANIM only) – String that specifies the message to deliver when the character reaches its destination **passOverMessage** (for RUN_ANIM only) – String that specifies the message to deliver if the character passes over another character **duration** (for RUN_ANIM only) – integer that specifies the duration for the animation to take **targetAbsoluteX** (for MOVE_TO only) – integer giving the x coordinate for the character to move to **targetAbsoluteY** (for MOVE_TO only) – integer giving the y coordinate for the character to move to **targetOffsetX** (for MOVE_INC only) – integer giving the x offset for the character to move **targetOffsetY** (for MOVE_INC only) – integer giving the y offset for the character to move	
<button>	**name** – string giving the name that this button will be referred to by and what will	ssuigamefsmspec

	be displayed on the button itself	

Grading

You should carefully document your code (including JavaDoc comments) and thoroughly test it. For grading it will be run against test cases unknown to you. These will be designed to uncover bugs in corner cases, missing validation of parameters, and other cases that might strike you as unusual.

If your code provides the elements described above and passes all tests you will receive 90/100 points (roughly a low A). If you wish to receive the remaining points you need to create an XML specification for an interesting or cool game idea. Grading for these extra points will be done on a competitive basis using a set of equivalence classes with the "best" category receiving 10 points, "good" receiving 8 points, etc. A selection of the better "extras" may be shown in class. If you don't want your work recognized in this way, please let us know.

What to Turn In

You will submit a zip file of your project directory The program is due Wednesday, October 22, 2014 at 11:59 pm. You should turn in your assignment on Blackboard by going to the assignment link, attaching a zip file with the contents below, and pressing SUBMIT. Please write a README.txt file which mentions any online sources you used to help with the project, as well as any notes about your project (i.e. if you couldn't get a particular part of the project to work). Create a zip file that contains the entire program directory from your workspace, and the README.txt file. Then, name the file p3_LASTNAME_FIRSTNAME.zip. For example:

p3_taylor_brandon.zip

Once you have made this zip file, go to the assignment page on blackboard, attach this file, and submit.