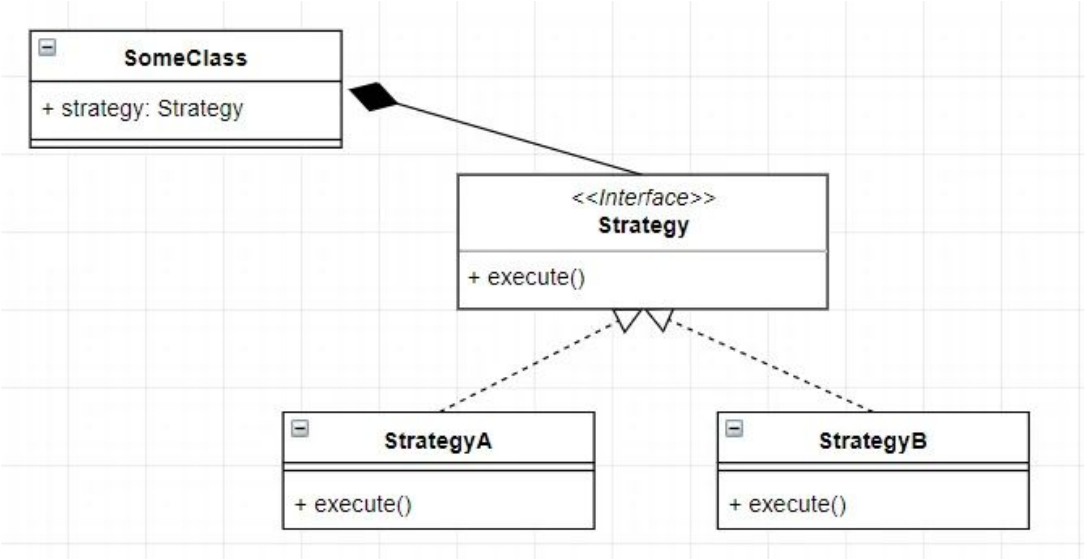
MVC
So the idea with mvc is that model view and controller are separate and hence can be tested independently of one and other
So the flow of information is from right to left (see uml). The controller tells the model what has been imputed and the model uses that input to modify the model of the game. The model is also responsible for editing itself over time as games change not just from player input. The model then passes information on to the view which receives a list of things (usually objects) all of which (some possible exceptions e.g. background) know how to draw themselves.

Strategy patterns
So the current plan is to make use of some strategy pattern probably one or two in model once we nail down more of the core mechanics and definitely one in view



So the idea with strategy patterns is that a given object has a strategy so it inherits the execute method; by giving it subclasses we can override the execute method. This allows us to avoid clumsy statement s like
if (gameObject == instanceOf(fireEngine)){
drawFireEngine();
}
insead we can simply call
gameObject.draw();
It also makes the code more expandable as you add new objects other classes can handle the exact same way they would handle other gameObjects

There is probably some more good practice and style stuff we should do EG functional programming, il try and weave that in as we go since we need to talk more before we nail some stuff down, also brand new language and all

Controler

Controller -

Responsible for a bunch of listeners (will need to read up on touch screen listeners)
Lets the model know what's been clicked where and what's been released

Model

Game-

Essentially the master class, holds a lot of privileged information. The game class owns the instance of player and all the game objects. The game class pulls together a lot of the logic of the game and controls timing. NB all the operations it performs on objects are methods native to the objects themselves.

Player-
Tracks points and decisions and such, functions would probably be fairly minimal
Getters setters and constructors, player for this game is fairly simple

gameObject-
Is the superclass that defines all game objects probably just a few basic bits
It should know where it is and how big it is. but it also defines all objects under it as gameObjects. Boolean draw, int draw priority bounding box, rasta array

orderItem-
Defines the commonalities with items found inside delivery boxes draw Method and other ways to manipulate objects

Item-
Placeholder name the actual objects inside the box themselves
For testing will want some predefined objects but long term will
Perhaps need someway to randomly generate from a large pool

-box
Defines the object that holds several objects and is passed to a player to sort through

-dialogueBox
Is this the best way to handle this ? but idea is text boxes that know how to draw themselves and can be constructed

Proceduralgenerator-
The idea is to offload the random elements of the game to another class
Should have methods to generate random events between rounds and make new boxes inside rounds

View

view -

Like game is the models master class view is the views master class. Holds a priority queue that determines what gets drawn ie background will always be priority 0 and drawn at the lowest level other objects drawn on top (this may not be the best way to do this open to ideas)

-cinematicView

What the player sees between rounds as such its handled differently

-gameView

How the view class can display all the content in the rounds

-mainMenuView

Can be greatly simplified or nonexistent in early builds and only becomes necessary later with increased functionality

Draw-

Interface

Is the outer strategy layer for drawing objects

drawObjectX-

Associates the object with a given image resource as the object.draw calls this passing its size and position over it doesnt know what it looks like

Visual resource-

A folder of png files for the program to draw on in order to show them on screen

They need to be png's or similar so we can use bounding boxes but also for clicking use a more sophisticated rasta array so the clickable part of the image is just the image not the rectangle its contained in