Aleksander Gisvold

# TDT4240: Patterns

**Step 1:**
- I choose the Pong game from Exercise 1.

**Step 2:**
- I already implemented the singleton pattern to my code. I have some coding experience, and have read about the singleton pattern on beforehand. Therefore I implemented it without the knowledge of this exercise.

**How I implemented the singleton pattern:**
- The ball as it's own class. Since there should only be one ball on the court at the time, you have to choices (at least from what I know). The ball may be a singleton instance sprite marked with 'Singleton' as a part of the game state, or the ball may be a separate class which is only used once in the game state class.
- I may implement the Singleton pattern to a greater degree, by making the game paddles into separate classes. I found this a bad idea since they reuse so much code, and it was easier to just make a method which only instantiates the paddles if the paddles do not exist already.

**Step 3:**
- I chose to implement the Model View Controller pattern.
It is a fairly understandable pattern and is already a part of the Android kernel. The Android libraries encourages use of MVC. The Observer pattern is a sub part of the MVC Pattern, so I will not describe how I implemented the Observer pattern, as this was not a focus, but rather a bonus.

How I implemented the MVC Pattern:
- As I understand, the MVC pattern is a part of the Sheep library too, making the use of MVC even more encouraged.
- The game world (new World()) is the model used to store information about score and sprites.

- The view is the display (Display display) that shows all graphical elements on screen. This is parted into a game layer and a collision layer. The game layer takes care of the representation of the movement and representation of sprites, while the collision layer detects collision.
- Android solves the Controller/View relation on a different way than theoretic MVC. Since there is touch controls on an Android device, the controller and view will be implemented together, in a way. I will explain this in the next section
- The game over state has a different view from the game state. This change of views is enforced by the controller, through an if-statement in the update() method.

- The touchlisteners are controllers moving responding to the touch events from the view. The controller will use methods such as collided(), onTouchMove(), update() and draw() to manipulate the view responding to whatever event is received.
- The controller manipulates the model through update(), e.g.

```
if(ball.getY()>=(canvasHeight-ballImage.getHeight())) //ball leaves court
```

```
 ball.setPosition(display.~~getWidth()~~/2, display.~~getHeight()~~/2); //controller
        ball.setYSpeed(-ball.getSpeed().getY()); //places ball in place in view
                      player2Score++;//controller updates model
```

**Step 4:**

a) Architectural patterns:
- Observer
- MVC

Design patterns:
- State pattern
- Template pattern
- Abstract Factory
- Pipe and filter

I was somewhat confused when solving this section. Different sources gave different arguments to whether the given pattern was architectural or a design pattern, and some sources were inconclusive. I have sorted them based on my understanding of the patterns.

Architectural patterns are the tools at a higher level, eg. what classes is the system parted into. Design patterns are the tools at a lower level, such as how do the classes interact, programming paradigms are the tools at the implementation level.

b) Answered in part 3.

c) MVC is a part of the Android kernel, and is therefore an encouraged pattern for Android applications. It is therefore easy and understandable to use in the context of Pong. Using MVC makes the code more understandable and easier to modify.
I see no disadvantages to MVC in this application.

(*)(*)
( - . - )
(@)(@)