Capstone1 Game

Content

Lab Report

Project Evolution

- At least half of the team was in presenting themselves in the teamroom, which is quite good. Others joined later.
- We started discussing an initial UI design of the game and corrected it to be more Pong alike.
- Another team member provided a git repository with a clone of the project. We started uploading the initial code there and used it throughout the course.
- An initial UML diagram was presented, but we never had a real discussion on it.
- From that time on, many members realized that they had not planned sufficient time for the project and they just stopped to respond
- I finished the project on my own, as I was interested in this project and wanted to refresh my old Java Knowledge. Thus the result of the project is mainly influenced by what I was interested in. I came up with a single and 2 player mode which are quite distinct to see if a flexible design can cover both. I also was interested in more complicated collision detection, the reason why there are rectangles that are not aligned to the X and Y Axis.

Problems

We came across the following problems:

- It took a while till we realized, that we needed to clone the github project, as we did not have the right to create our own branch in the original project.
- We did not find functions in the Processing Library helping to calculate a collision between a rectangle and a ball. Thus we needed to search the net for it, but soon found solutions.
- For me the MVC pattern does not match the given problem. The lessons of the course have not been helpful here. The multiple options and changes just made it more confusing. Thus I decided to go without it. See also the discussion in the design part.
- The initial decision to implement a 2 player game made it hard to test for a single person. Because of this I came up with a single player mode.
- Also using a keyboard driven paddle is very hard to play.
- Using Java 1.8 is quite old and Java 10 allows for better solutions. I sometimes got stuck as my solution dis not work until I realized, that it is not supported with Java 1.8

How to play the game

This game provides a 2 Player mode and a single player mode. If started the game immediately starts in 2 player mode. The mode can be switched by following keys:

- “1” for single player mode
- “2” for 2 player mode.

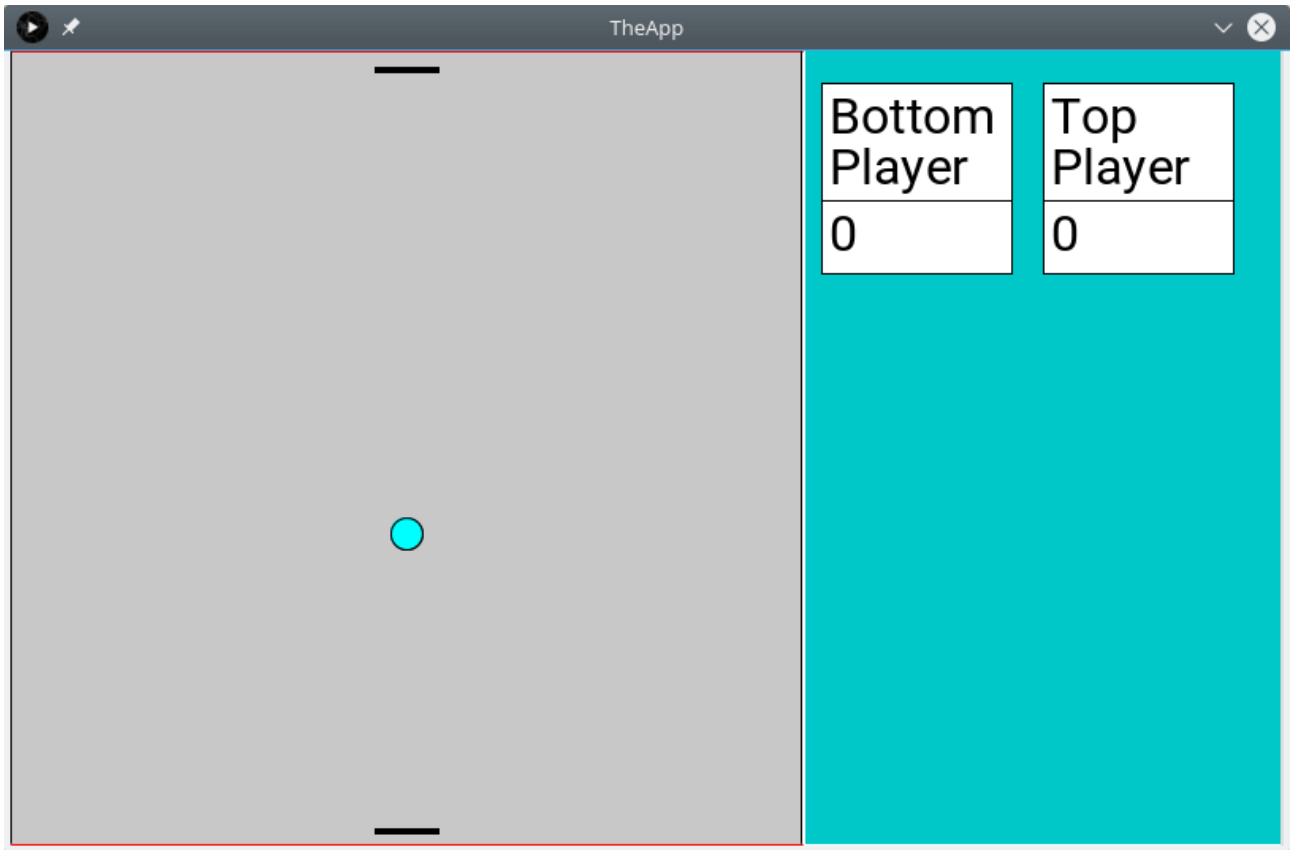
The counters are not reset6 when switching the mode, but this can be achieved by pressing the following key:

- “r” for resetting the counters

2 Player Mode

In 2 player mode you will see a playground with 2 black paddles. One on the top and one on the bottom. The paddles can be moved to the right and left using the keyboard and also can be tilt. A paddle that is tilt will reflect the ball in a slightly different direction than the paddle in the normal position.

The top and bottom border of the playground are marked red to indicate that in case the ball hits these borders, the other player will get a point. Hitting this border will increase the score. The side borders are marked by a black line and will reflect the ball.



The bottom paddle can be controlled by the keys a s d f w e and the top paddle by the keys k l ö ä p o. This assume a keyboard with a German layout.

Key left Paddle	**Key right paddle**	**Action**
a	k	Move left fast
s	l	Move left slow
d	ö	Move right slow
f	ä	Move right fast
w	p	Tilt left
e	o	Tilt right

To change the keys, please change the line 20 and 21 in the class Game.

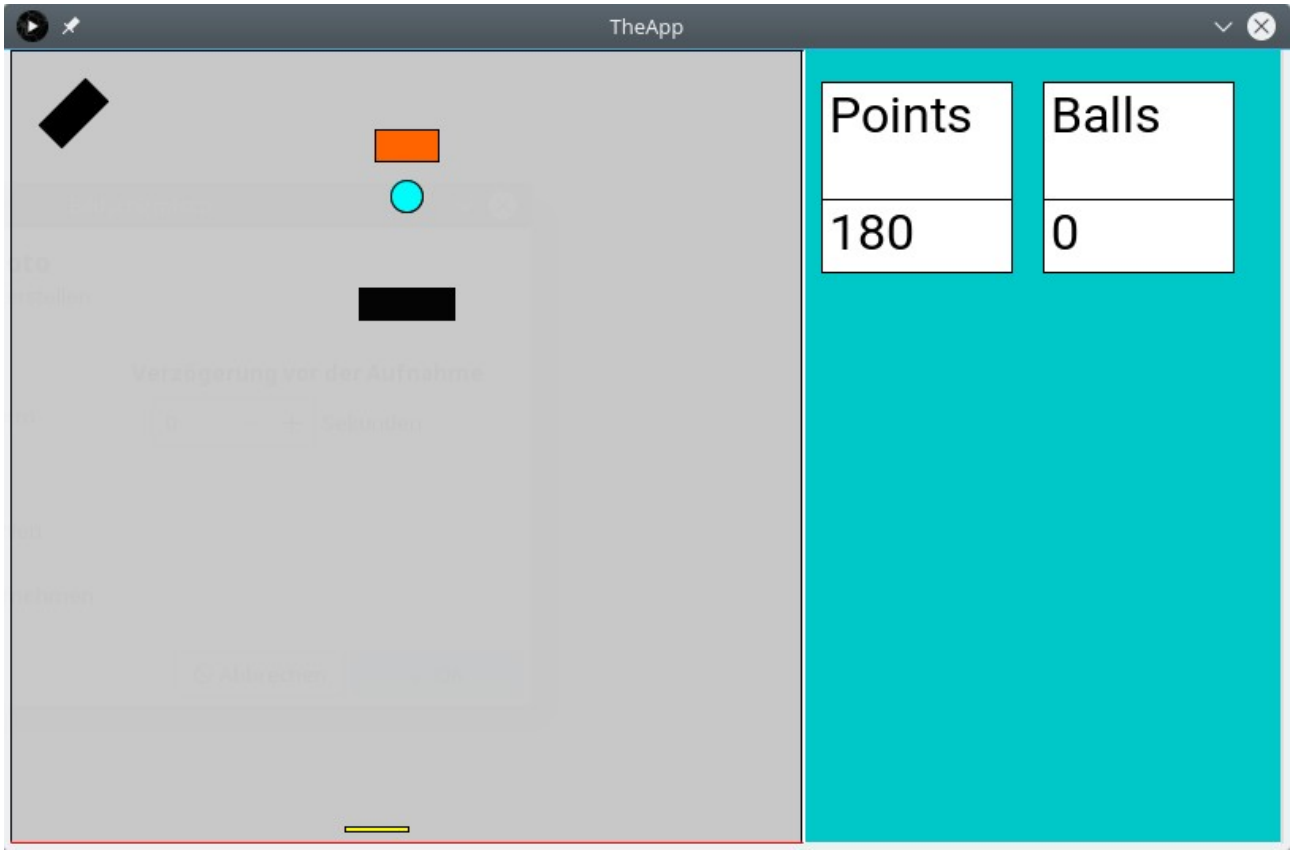
In case the ball hits the red border it gets invisible, as this ball is lost. To get a new ball press the key:

- “b” to get a new ball.

This will put a new ball int6o the middle of the game and start it in direction to the bottom. There is some variation into the direction from the second ball on.

Single Player Mode

Below the game in single player mode.



This mode of the game is a variation of the Squash game. This time we are using the mouse to control the yellow paddle. In case the mouse pointer is inside the game, the x position of the pointer will determine the location of the paddle. The left and right mouse keys can be used to tilt the paddle.

- Mouse location controls the paddle x location
- Left mouse key = tilt left
- Right mouse key = tilt right

Also only the bottom border will catch the ball all 3 others will reflect the ball.

Points are collected by hitting the red rectangle with the ball:

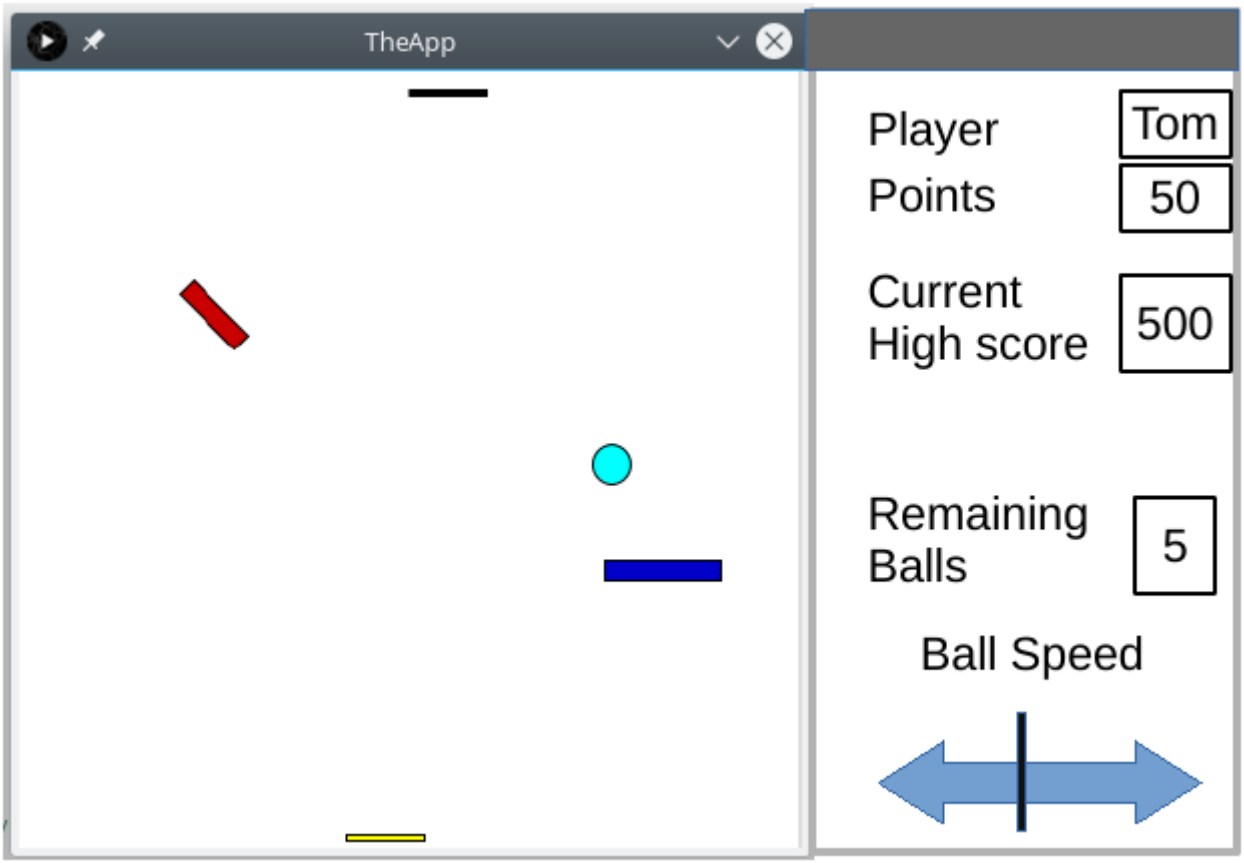
- 5 points for hitting it on the long side
- 20 points for hitting it

The goal of the game is to get as many as possible points with a given number of balls

Design

UI Design

Here our initial UI design with a black keyboard paddle and a yellow mouse paddle and 2 obstacles changing the direction of the ball



The design has been changed for the following reasons:

- The mouse paddle has been replaced by a keyboard paddle to have only 2 keyboard paddles. As the keyboard paddle is much harder to play as the mouse paddle, the game would be unfair otherwise. We also added a fast and a slow mode for the keyboard paddle, to make it easier to play. The keys for the movement have been selected from left to right and the keys for tilting are the 2 in the middle above the move row.
- The obstacles have been removed for the 2 player mode to fit the pong requirements.
- The counter for the player has been replaced by 2 counters, as the initial design was wrong . One for each player.
- High score is not yet implemented

- Remaining Balls needs to be implemented only in the single player mode, but is not yet fully implemented. There is only a counter counting the balls.
- Ball Speed was meant to increase or decrease the ball speed. This also would have to influence the score in case of the single player mode. Also not yet implemented.

Code Design

We looked at the following design pattern to implement the game:

- MVC
- Observer
- Strategy

Observer and Strategy have been fitting quite well, but MVC is only partially implemented.

Using MVC or not

We found it quite natural, to implement the drawing objects in a way, that the objects could draw themselves on the Papplet. By implementing this, the view and the model is more or less collapsing. We think that full MVC is not really fitting because:

- The objects we are modeling are drawing objects which hold basically not more than their information on how to present them on the screen.
- In the multi player mode there is only a single view, which is always presenting the full information of the model. MVC make much more sense when multiple views exits presenting only parts of the model.

Adding the single player mode adds more or less a second view and not all the objects are shown all the time. Thus MVC would now make more sense, but time does not permit to fully restructure the code.

The Model of the Drawing Objects

The model of our game is composed of drawing artifacts such as ball and rectangle, that have all in common:

- All of them have a position on the screen
- they may be visible or invisible
- they have a color
- may be rotated by an angle
- can be drawn on the Papplet
- may collide with the PongBall
- and hold other drawing artifacts inside (for example the paddles are inside the playground)

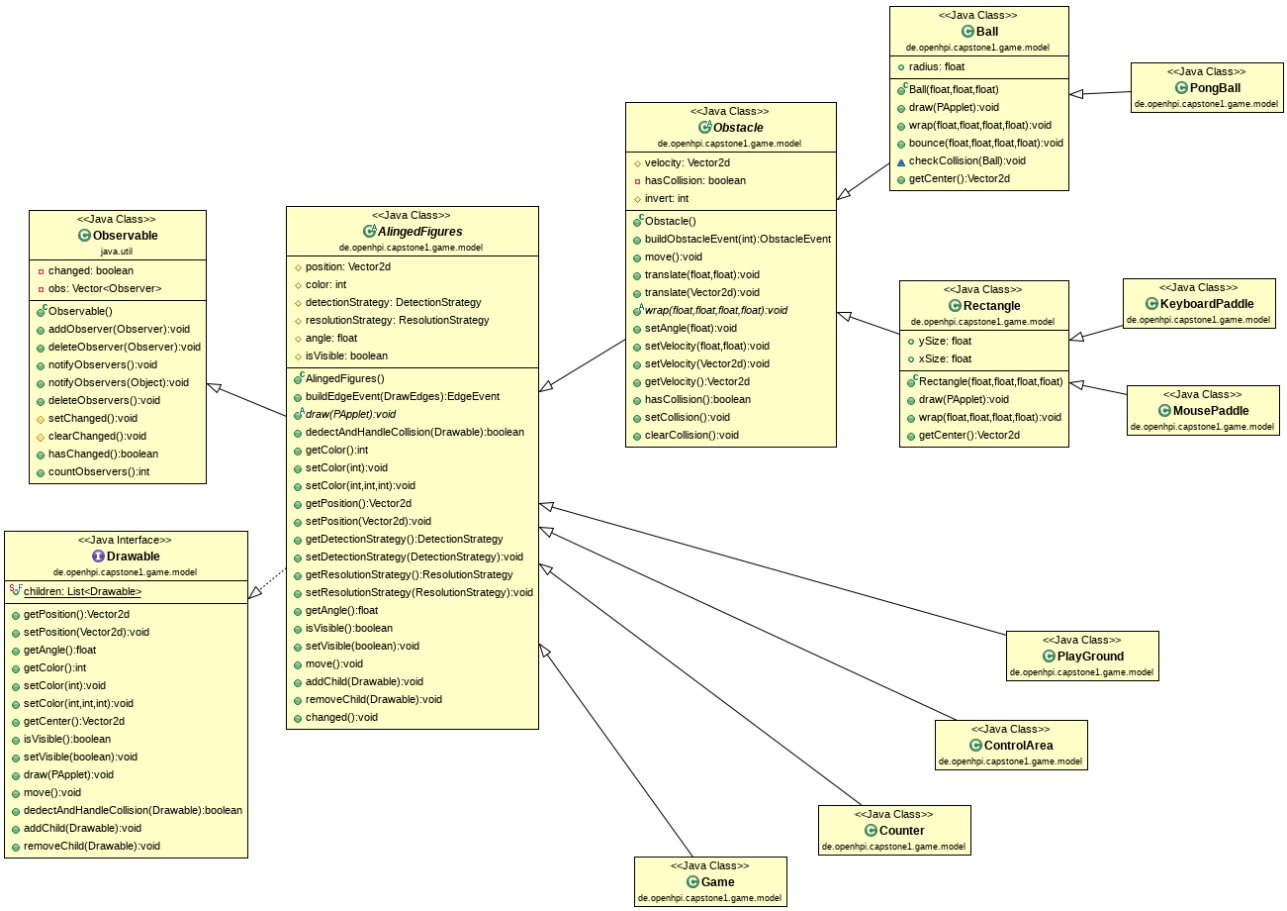
All needed methods to handle these common attributes and behavior are summed up in the interface “Drawable” which is implemented by all of the drawing objects.

As controllers may be interested on some state of the Drawables, they also extend the Observable class. This enables Observers to register for state changes.

On top of this interface and the base class following Drawables are implemented:

- The abstract class AlignedFigures implementing most of the methods of Drawable for objects that can not move and are alligned to the X and Y axis.
- Game, Counter, PlayGround and ControlArea implement this class
- The Class Obstacle extends AlingedFigures and adds the possibility to rotate the object and to give it a velocity.
- Ball extends Obstacle and is the base class for the PongBall
- Rectangle also extends Obstacle and is the base class for Mouse and KeyboardPaddle.

The full class diagram including the inheritance dependencies is shown below.



Vector2d

As there was the need to handle 2 dimensional coordinates and operations on them, we came up with the class Vector2d representing a location or velocity in the 2 dimensional Space. Following actions are supported:

- adding 2 vectors
- subtracting 2 vectors
- multiply a Vector by a factor
- normalize a Vector (keep direction but resize it to the length of 1)
- rotate a Vector

Using the Observer Pattern

We used the Observer pattern multiple times in our Design, especially for the controllers. Lets look specifically into following usage:

- In the single player mode, the red rectangle is hit by the ball, which needs to increase the counter.

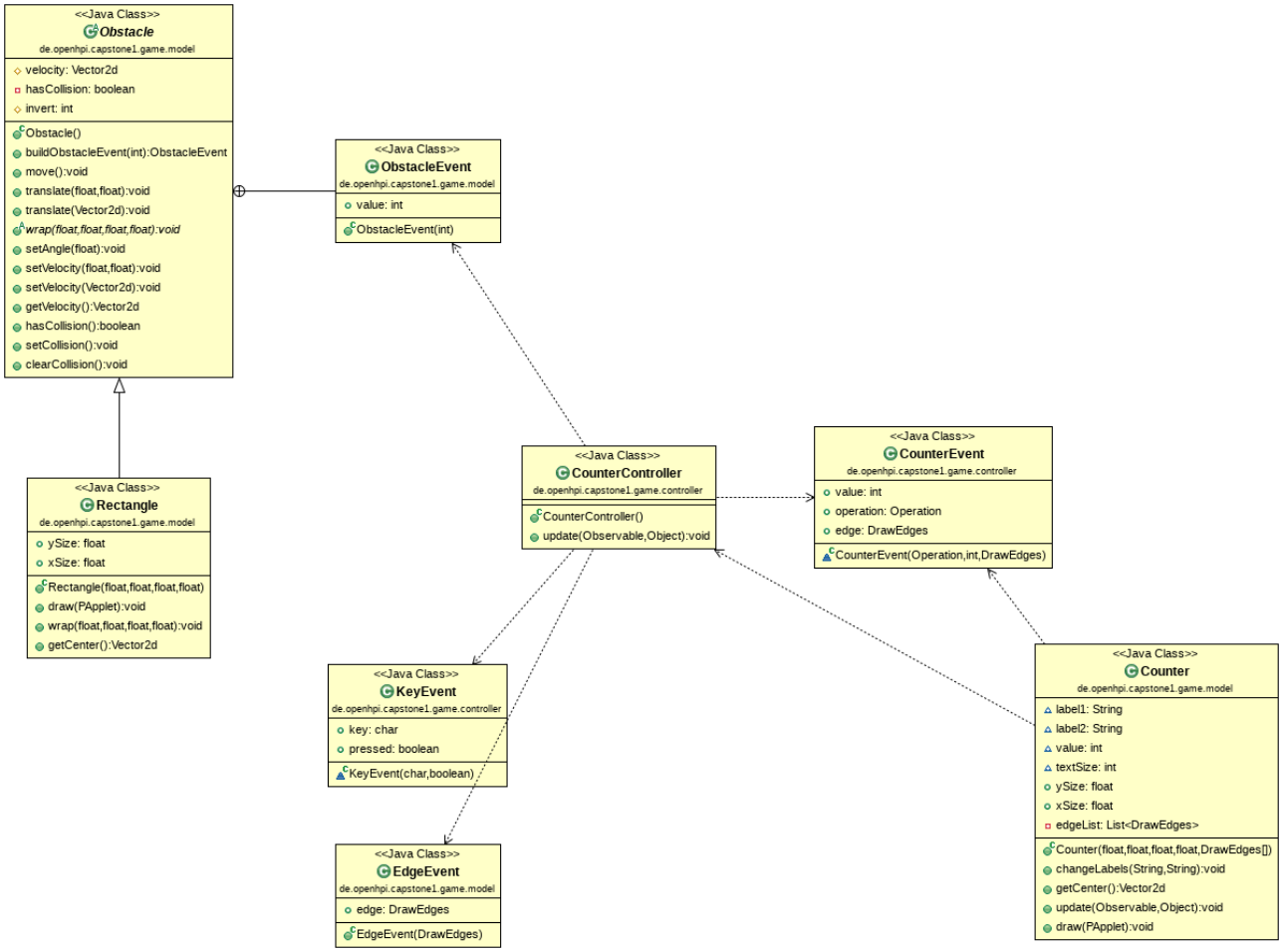
Following Observer – Observable interactions are now coming into the game:

1. Because the CounterController has registerd as Observer at the red rectangle, the rectangle creates an ObstacleEvent indicating if the small or large side was hit and sends it to the CounterController.
2. Because the Counters (there are 2 of them) have registered as Observer on the CounterController, the counter Controller sends a CounterEvent to the Counters indicating if the Counter should be increased, decreased or reset and information of the origin to the counter. Based on the origin, the counters decide if the event is for them or not.

As can be seen in the example above, the CounterController acts in this case as Observer and as Observable. We used the core java classes to realize this behavior.

Next to the Obstacle event, the CounterController also registered for KeyEvents to handle the character “r” to reset the counter and on the EdgeEvent to handle a hit of the ball on the red edge of the playground.

The following UML diagram focuses on the dependencies between the classes involved in the example above.

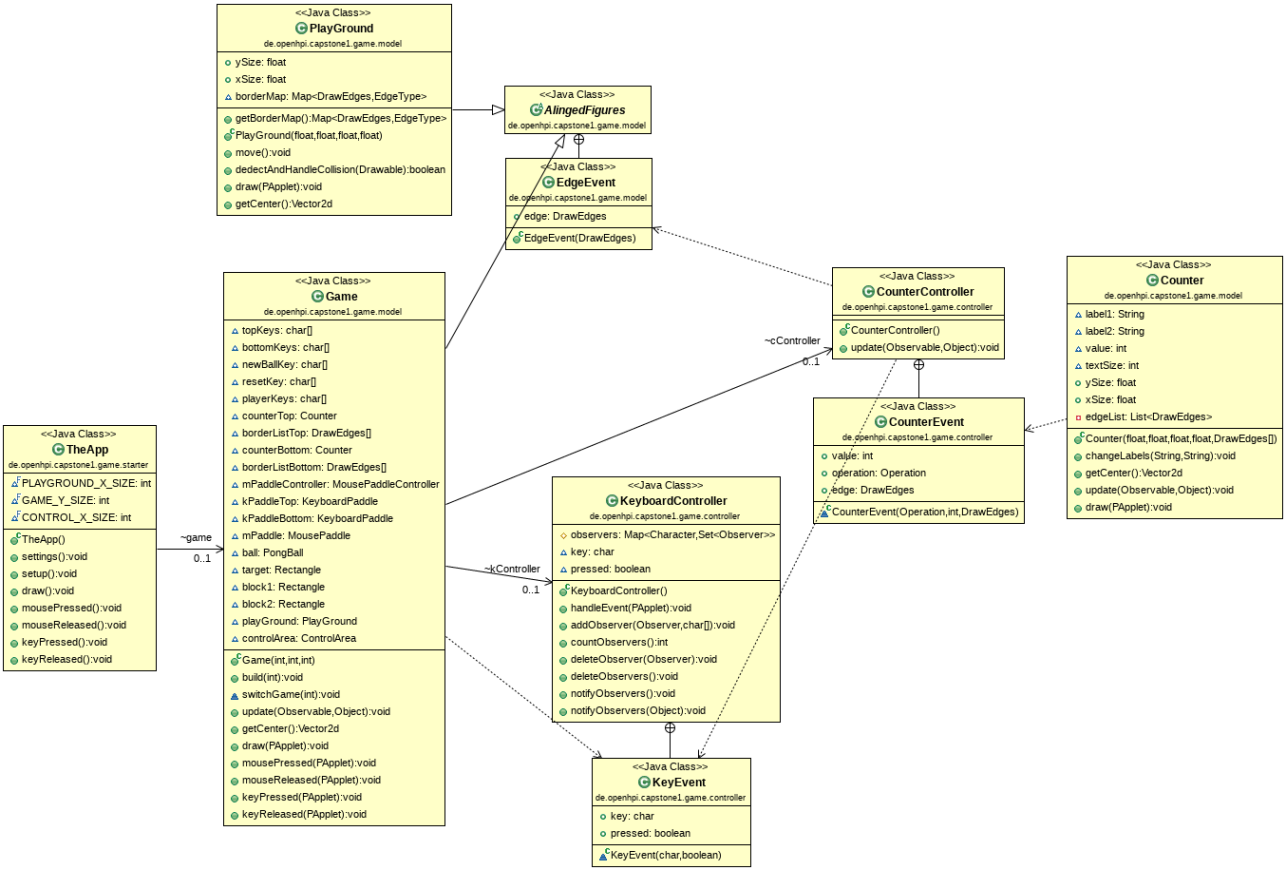


The following events are exchanged between the objects of the game:

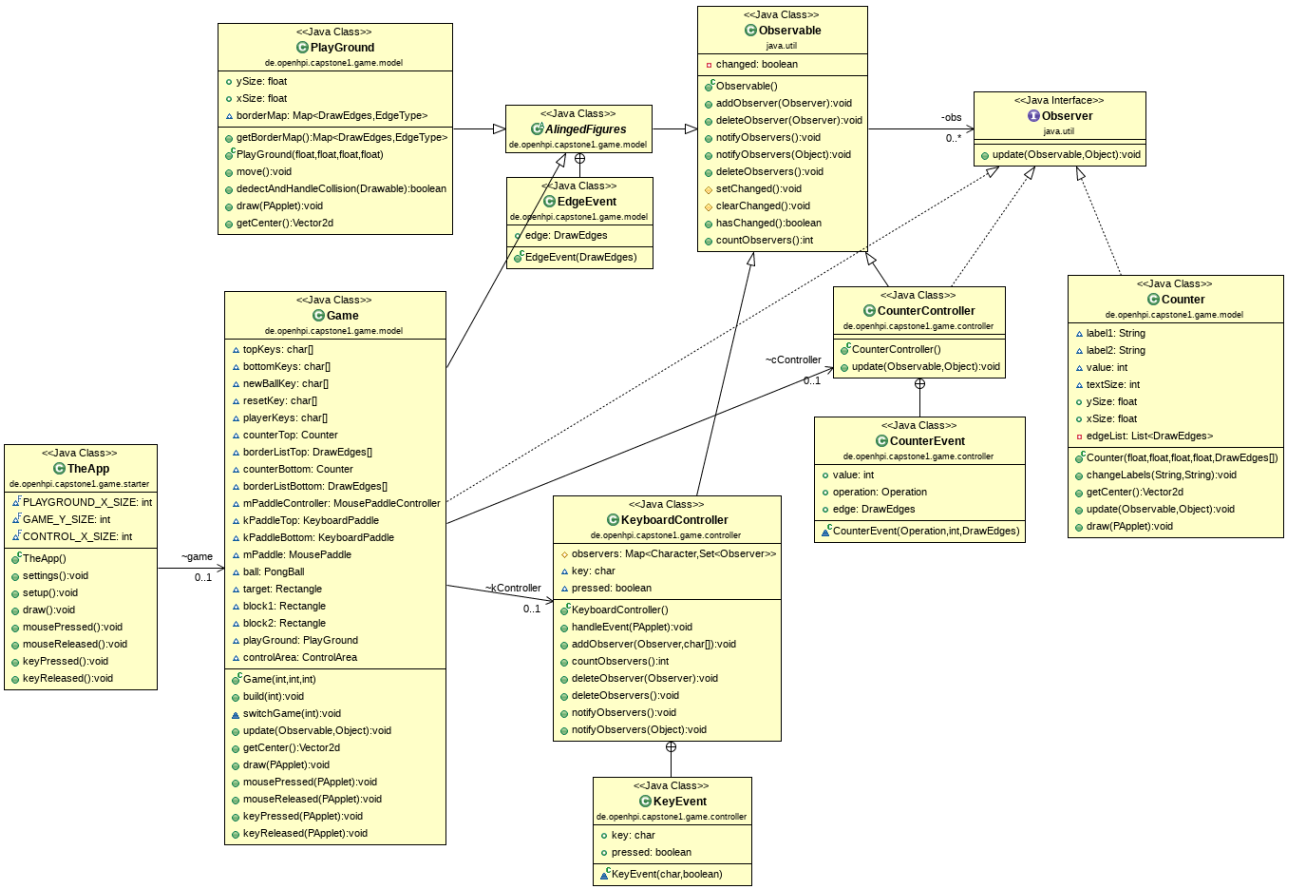
- Mouse and Keyboard events are exchanged between the App and the Game. This interaction is not realized by the Observer pattern, but is hard coded.
- Also the interaction between Game and the KeyboardController and the Game and the MouseController are hard coded, as we did not have the need for alternative solutions here.
- Between KeyboardController and the registered Observers CounterController, Game, and KeyboardPaddle the Observer pattern is used.
- Also the Counter controller needs to react on multiple events, thus it registers at the red rectangle the keyboard controller and the Playground.
- The Counters themselves register as Observer at the CounterController.

In summary, we used the Observer pattern whenever we have multiple parties interested in the events or there are multiple sources of events, potentially changing between modes of the game. Whenever there is a constant 1:1 relationship between the event generator and the consumer, we used “hard coded” forwarding of events by directly calling the method of the receiver.

Below the relationships between the event generating and consuming classes of the 2 player game.



To realize the Observer pattern, different classes need to implement the Observer interface and inherit from the Observable class. This inheritance is shown below.



Using the Strategy Pattern

All the drawable Objects are using 2 Strategies, which can be plugged into the object:

1. One strategy to detect if there was a collision with the ball
2. One strategy for handling the collision

Collision Detection

The implemented strategy for the collision detection works, but may have a drastic performance impact, in the case many objects are placed on the play ground. In this case another strategy may be needed. This 2 step strategy will first check if the distance between the center of the rectangle and the ball will make a collision possible at all. If the ball is near enough the more performance intensive calculation will be use.

Another reason for changing this strategy is given in the case of the Space Invaders game. Here the Ball has such a small radius, that it can be neglected in the calculation. This will also increase the speed of the calculation. This may be required, as we have mac balls in this game.

Collision Handling

As can be seen in the single player implementation, some rectangles just reflect the ball, others are also able to emit events. Replacing the strategy also allows to:

- destroy the rectangle in case it is hit
- change properties of the ball, such as its velocity
- and much more

Hit Detection

Finding out, that the ball hit an object or the border is quite an important aspect of the game. We needed to know:

- that the ball hits one of the boarders of the play ground
- that the ball hits the paddle or a rectangle

Boarder

The boarder is quite easy, as one just has to find out thet the distance between the ball and the boarder is less than the radius of the ball. As the playground is alligned to the X and Y axis, an easy task. It just has to be done for each of the 4 boarders.

Rectangle

We decided to model the paddle as a rectangle, as we wanted to use the current implementation also as a base for implementing the other games where rectangles are needed. As we did not find something in the Processing Library, we checked the internet and found a good article on this topic:

https://stackoverflow.com/questions/401847/circle-rectangle-collision-detection-intersection

We implemented/copied one of the proposals and modified it, as we wanted to support rotated rectangles as well. Now we have the following approach:

- rotate the coordinate system to align the rectangle with the X and Y axis
- Use the initial approach to find out if the rectangle is hit
- calculate the new direction of the ball in case of a collision
- rotate back to the original coordinate system.

The detection and the handling of the collisions works quite well, but I had at least on situation where it looked like the ball stuck inside the rectangle for 2 or 3 collisions. As this is not stopping the game, just the reflection was not done correctly, we ignored it for the moment.

Testing

Automatic testing of the classes was not required for this work, but we still did some Unit Tests which could be found under the source folder „test“ to make sure 2 base classes are working well:

1. There are tests for the Vector2d class to make sure that it works correct as it will be used in many calculations
2. and tests for the class Keyboard controller. As this class implemented a observer registry for each character, we wanted to test this class more thorowly.

The unit tests have been implemented using Junit and we are using the build in support inside Eclipse to run them